The Basics of *BASIC*

Apple Version

Created, written and programmed by:

NEIL RADER

SCOT ROBINSON

Project supervisor:

BARRY KASVEN

Manual written and edited by:

BARRY KASVEN

SCOT ROBINSON

NEIL RADER

DREW WEINLANDT

839 Stewart Avenue

Garden City, New York 11530

(516) 794-8900

PREFACE

Your Apple Computer is an extremely useful tool. The Basics of BASIC is an aid to help you use your computer as well as help you understand how it works. Basically, you have two options when using your computer:

1) You may BUY existing programs
2) You may WRITE your own programs

One of the things that makes your Apple so popular is the availability of programs that can be used on it. While many of these programs are good, they are not always exactly what you are looking for. For this reason you may decide that you would like to write your own programs. The Basics of BASIC is an interactive aid to teach you programming.

By using this interactive system you should learn the fundamentals of BASIC programming. When you complete all of the computerized lessons and the manual you will be skilled at writing computer programs. Each lesson will cover a specific topic in detail and then drill you with exercises. The best part is that you will be using the computer to help learn about programming. In this way, you can try things out as you learn them, as well as become familiar with the computer system.

SCOT ROBINSON
NEIL RADER

TABLE OF CONTENTS

INTRODUCTION

Welcome to the Basics of BASIC! If you're new to the world of computers and computer programming, then our set of 12 lessons will introduce you not only to the BASIC programming language but to the APPLE computer as well. If you are using this package as a refresher, you'll find that the lessons are simple enough so that you won't need extensive preparation in order to understand them.

Since you're obviously reading the manual at this time, let's talk a bit about how these lessons are intended to be used. Each concept to be learned is first introduced on the diskette. Diskettes are self-booting (that is, they load their own instructions automatically when you turn the computer on and place the diskette in the disk drive), and self-explanatory. Instructions for use are displayed on each screen as the program progresses, so that no experience in using the computer is necessary in order to proceed from screen to screen or from program to program.

Lessons on the diskette are presented in a straightforward manner in plain English. Jargon- or "computerese"- is held to a minimum, and interaction with the computer is maximized. Lessons are arranged sequentially. Concepts are carried forward as they are developed and practiced, and they are applied and used in succeeding lessons in a manner designed both to reinforce them and to suggest the variety of their applications.

This manual is intended as a supplement to the lessons on the diskettes. It is designed to reinforce material already presented and to expand upon it. The Table of Contents indicates each topic discussed within a chapter (and within the corresponding lesson on the disk).

Each chapter within the manual, which corresponds to a lesson of the same number and name on the diskettes, is divided into three parts. Part one, entitled "What Have You Learned?", presents a broad overview of the major new ideas developed within the lesson. Part two, called "Let's Take a Closer Look" explains each idea in detail and illustrates it where applicable using programming examples. Part 3 is a review and practice section with questions and exercises appropriate to the lesson. Chapters four through twelve also contain additional practice problems with their suggested solutions.

The manual concludes with a glossary of important terms and Applesoft BASIC commands and two appendices, one that introduces important terminology related to computer hardware and one that presents a step-by-step method for formatting a personal work disk.

If you follow the instructions on the diskettes and use this manual in conjunction with them, you will be capable of simple programming by lesson 4, and you will be able to write reasonably complex, useful programs by the time you have completed the series. Good luck to you as you embark upon your journey into the world of computers!

GETTING READY

Before you begin your study of "The Basics of BASIC," let's take some time to look at a few procedures that will make your use of this series on the Apple II, Apple II Plus and Apple IIe more enjoyable and fulfilling.

(1) CONTROL OF SCREEN PRINTING SPEED AND SOUND:

The ESC key on your Apple keyboard will allow you to a) change the speed at which letters are printed on the screen and b) turn the sound on or off. The ESC key will generally (though not always) work when the computer is waiting for you to press the RETURN key.

When it does work, the following information will appear on the screen:

```
CURRENT SPEED = 6
PRESS <RETURN> TO KEEP SAME SPEED
1-FASTEST      9-SLOWEST
ENTER NEW SPEED (1-9)>
```

You must either enter a new speed and press the RETURN key or simply press the RETURN key. In either case, you will then see the following message:

```
DO YOU WANT SOUND (Y/N)
```

You may press RETURN to continue the previous status or press the "Y" or "N" keys to answer "yes" or "no".

On those occasions when the ESC key does not work, your status will be unaffected. You will hear a noise- the computer's way of telling you that it has not carried out this instruction- and you may simply continue until another opportunity arises to press the ESC key.

(2) HOW TO GET OUT OF A LESSON:

There may be occasions during your progress through these programs when you will want to break out of a lesson. You may, for instance, find that you do not understand a concept and you may want to return to the beginning of the lesson to re-run it, or you may find that you are working on a lesson in which the concepts are already familiar to you.

If so, holding down the CTRL key while pressing the letter 'C' will cause the following to appear on the screen:

```
PROGRAM HALTED
ENTER YOUR CHOICE (1-2)>>
1. RE-RUN THIS LESSON
2. RETURN TO MAIN MENU
```

If this message does not appear, press the RETURN key and then repeat the above procedure.

(3) WHEN YOU'RE FINISHED:

Always follow these four steps when you've completed your work and you're not going to be using the computer for a while:

1. Wait until the red light on the disk drive is off.
2. Open the door of the disk drive.
3. Carefully remove the disk.
4. Turn off the computer.

(4) A DISK DIRECTORY:

DISK #	LESSON #	TITLE
1	1	An Introduction To The Keyboard
	2	All About Printing in Direct Mode
	3	All About Strings and Variables
2	4	An Introduction To A Program
	5	A First Look At Input
	6	Adding Some Spice To Our Outputs
3	7	A Little Looping And Branching
	8	Learning To Make Decisions
	9	String Handling Techniques
4	10	The Integer and Random Functions
	11	Looking Into Arrays
	12	A Review of BASIC Error Messages

LESSON 1
AN INTRODUCTION TO THE KEYBOARD

WHAT HAVE YOU LEARNED?

Well, that was a good beginning! You can't start to learn about programming until you understand how to communicate with your computer.

You learned that you communicate with your computer with the help of its keyboard. You enter information and the computer tries to interpret it. So now you should know about the following:

(1) How to use the RETURN key;
(2) How the Apple's keyboard works;
(3) How to do some simple line editing using the left and right arrow keys.

LET'S TAKE A CLOSER LOOK:

(1) How to use the RETURN key:

The RETURN key signals the computer that you are FINISHED ENTERING INFORMATION and you are waiting for a response. After all, how can the computer know that you have completed entering information unless you tell it so!

It is essential that you press the RETURN key when you are finished typing information. As you progressed through Lesson 1 on the computer, you were constantly pressing the RETURN key to signal the computer that you were finishedreading each screen. You also used the RETURN key to let the computer know that you had typed in a response to various questions.

How did you know it was all right to press the RETURN key? You found out that when the computer is finishedprocessing data it signals you with a small blinking line called the CURSOR. Actually, the cursor may take many forms, including a blinking block, a blinking line, or even a non-blinking block or line. The appearance of the CURSOR is the computer's way of letting you know that it is your turn to do something- perhaps to press RETURN or to indicate a response.

(2) The Apple's Keyboard:

You learned that, although they look similar, your Apple's keyboard differs from a typewriter's in a number of important

ways:

(1) One difference mentioned above is the RETURN key. Although a typewriter has a RETURN key, its purpose is simply to advance you to the next line of type.

(2) There are several keys that are interchangeable on a typewriter, but not so on your Apple:

(a) The "1" and "l": A typewriter accepts either of these as the number one, but your computer will not! You must press the number 1 and not the lower case "L".

(b) The "O" and "0": A typewriter doesn't care which you use but your computer does. As you learned in Lesson 1, the number "zero" has a slash through it (0) while the letter "O" does not.

(3) How To Do Some Simple Line Editing:

In this section you learned a simple way to make a correction if you accidentally pressed the wrong key. On your Apple, you can make a correction as long as you haven't yet pressed the RETURN key. Suppose, for example, you typed:

MY NAME IT NEAL

You really meant to type IS rather than IT. If you have not already pressed the RETURN key, you can correct the word by using the left and right arrow keys.

Everybody makes mistakes and you learned that the computer lets you correct your mistakes easily and quickly without having to re-type text.

The LEFT ARROW KEY is used to move the CURSOR to the left when you want to change a letter. As it moves over each letter it doesn't change the letter itself. You change the letter yourself, if you recall, by simply typing the correct letter or number over the incorrect one.

The RIGHT ARROW key is then used to move the cursor to the right over existing letters without changing them. The computer only recognizes information to the left of the cursor, so using the right arrow key to pass over letters is a shortcut for retyping them.

If you need further practice with any of these techniques, it would be best for you to repeat Lesson 1. You will see that the only way to become a really good programmer is to practice.

EXTRA INFORMATION

In the following section we will discuss additional keys that are important to learn before beginning to program:

(A) SHIFT
(B) REPT
(C) ESC
(D) CTRL

(A) SHIFT:

Your Apple's shift key has a purpose similar to a typewriter's. On an Apple II and II Plus, however, there are usually no lower case letters available on the keyboard, so the only purpose of the SHIFT key is to print these special characters: ! " # $ % & ' ( ) * + < > ?

These special characters are located on the top portion of various keys on your keyboard. By holding down the SHIFT key, for example, and pressing the number 3, you will see # displayed. The other symbols are displayed in a similar manner. If you have an APPLE IIe, however, the shift key can be used in conjunction with the alphabet keys to display upper case letters and these special characters:

& * ( ) ! + % " < > ? # $

The CAPS/LOCK key located at the bottom left on the keyboard can be depressed to toggle between upper and lower case. However, BASIC PROGRAMS WILL RECOGNIZE ONLY UPPER CASE LETTERS, SO YOU MUST MAKE SURE THAT THE CAPS/LOCK KEY IS SET FOR UPPER CASE WHEN USING THIS PROGRAM OR WRITING YOUR OWN PROGRAMS.

(B) REPT (ON APPLE II AND APPLE II PLUS ONLY):

REPT is short for the word "REPEAT." This key does just what it says: it will repeat any letter or character (or even a space) as long as you hold it down. For example, if you press the letter A while pressing the REPT key at the same time, the letter A will continue to print. If you are near a computer now, you can try it out on a few characters.

On the Apple IIe there is no REPT key. Any key will automatically repeat if it is held down for more than one second.

(C) ESC:

Don't worry about this one too much for now. The ESC key, which stands for "ESCAPE," has a variety of uses. In this program, for instance, it has been used to vary the speed with which words appear on the screen.

(D) CTRL (ON APPLE IIe, labelled CONTROL:

CTRL is short for "CONTROL" and, like the SHIFT key, it is always pressed together with another key. It has various functions which will be discussed as you learn more about your computer.

(Want to try a simple use of the CTRL key? If you are sitting at a computer, press down CTRL and press the "G" key. You should hear the Apple's bell ring. Your other hint is that the word "BELL" is located on the "G" key).

(E) RESET:

This key is located in the upper righthand corner of the Apple's keyboard. It is activated in conjunction with the CTRL key to break out of a program and return the user to the programming mode. It should NOT be pressed while using the Basics of BASIC since it will cause the computer to restart, thereby clearing whatever program is currently in use.

A CHANCE TO REVIEW AND TRY OUT YOUR KNOWLEDGE

I. TRUE-FALSE

1. The ESC key signals the computer that you have completed entering information.

2. The lower case "L" or the 1 may be used to represent the number one.

3. The name given to the small blinking line or square displayed by the computer is the cursor.

4. The computer does not distinguish between an "O" and a zero (0).

5. The left arrow key ( ) erases each character as it moves the cursor over it.

II. COMPLETION

1. To type the same character over and over again without having to press down the key each time, hold down the character's key and the key.

2. Two keys that must be used in conjunction with other keys are and .

3. To do line editing, the key will allow you to move the cursor from the end of a word towards the beginning of a word.

4. The zero looks different than the letter O in that the zero .

5. The key signals the computer that you are finished entering information.

ANSWERS

CHAPTER 1

TRUE-FALSE

1. FALSE. The RETURN key is used.
2. FALSE. Only the number 1 may be used.
3. TRUE.
4. FALSE. Any time the number zero is required, the Ø must be used.
5. FALSE. It passes over the character without erasing it

COMPLETION

1. REPT (This is only true for the Apple II and Apple II Plus. For the Apple IIE any character can be repeated by simply holding the key down for more than one second.)
2. CTRL and SHIFT
3. LEFT ARROW
4. has a slash through it
5. RETURN

LESSON 2
PROGRAMMING IN DIRECT MODE

WHAT HAVE YOU LEARNED?

If you've now gone through Lesson 2, then you can see that your Apple is a very useful tool. You were able to give it commands from the keyboard which it was immediately able to interpret (translate) and execute (perform). As you learned, this is called programming in the DIRECT (or IMMEDIATE) mode.

Your computer is able to recognize many statements which are stored in its own personal vocabulary. One very important statement that you have now practiced is PRINT.

LET'S TAKE A CLOSER LOOK:

(1) Printing A Literal:

The computer allows you to display any message on the screen. This function of the computer is called "printing a literal."

The way you go about printing a literal is by using the form:

```
PRINT "MESSAGE"
```

MESSAGE can be any word, words, or group of characters. Notice the double quotes, though. These are important. If you recall, all characters that you want printed by the computer must be placed between the quotation marks. (Quotation marks are located on the top part of the "2" key) on the Apple II and Apple II Plus and on the top part of the single quote (') on the Apple IIe).

You saw, for example, that you could enter the following line into the computer:

```
PRINT "THE BUS IS YELLOW"
```

The computer then did exactly what you told it to do. It printed:

THE BUS IS YELLOW

Your computer is able to do this because it recognizes the PRINT statement and displays whatever is between the quotation marks. Remember: only the characters BETWEEN the quotes are displayed- not the quotes themselves.

Here are a few other examples of PRINTing a literal:

If you type:	Your computer responds:
Print "I LOVE MY APPLE!"	I LOVE MY APPLE!
Print "THIS IS EASY SO FAR."	THIS IS EASY SO FAR.
Print "1 + 3 = 4"	1 + 3 = 4

(2) Formatting the Output:

After learning how to use the PRINT statement, you learned how to use two keys to format (or arrange) words or characters on the screen: the comma (,) and the semi-colon (;).

A. THE COMMA: The comma tells the computer to print each word or character in a separate column. This is especially useful for printing headings and tables. Here are some examples:

If you type:

PRINT "NAME", "ADDRESS", "AGE"

Your Computer Responds:

NAME ADDRESS AGE

If you type:

PRINT "COLUMN 1", "COLUMN 2", "COLUMN 3"

Your Computer Responds:

COLUMN 1 COLUMN 2 COLUMN 3

B. THE SEMI-COLON: the semicolon provides a way for you to tell your computer to put two words or characters right next to each other with no spaces in between.

You saw, for example, that if you typed:

```
PRINT "AUTO"; "MOBILE"
```

your computer printed:

```
AUTOMOBILE
```

Here are 2 more examples:

If you type:	Your computer responds:
`PRINT "COM"; "PUTER"`	COMPUTER
`PRINT "THIS"; "IS"; "A"; "TEST"`	THISISATEST

Although the use of the semi-colon in this way probably made no sense to you at first, that was only because you hadn't yet learned some additional programming techniques. One such technique is using your Apple as a calculator.

(3) Using Your Computer As A Calculator:

Besides having your computer print words, you also saw how easily it can be used as a calculator. Do you remember how?

Here are the arithmetic symbols that your computer recognizes:

THIS SYMBOL:	IS READ AS:	TO PERFORM:
+	plus	Addition
-	minus	Subtraction
*	times	Multiplication
/	divided by	Division
	to the power of	Exponentiation

In order to use the computer to do arithmetic, all you have to do is use the PRINT statement with the proper arithmetic symbol. Here are some examples:

Arithmetic problem:	You type in:	The computer responds:
Add 127 to 43	PRINT 127 + 43	170
Subtract 10 from 30	PRINT 30 - 10	20
Multiply 411 by 12	PRINT 411 * 12	4932
Divide 25 by 5	PRINT 25/5	5

Now, let's take a look at the semi-colon again. How does it help you to use the computer as a calculator? The answer is that it makes it possible for you to combine the PRINT statement with your calculations. Let's see how.

When you typed in:

```
PRINT "FIVE PLUS TWO IS "; 5 + 2
```

Your computer responded with:

```
FIVE PLUS TWO IS 7
```

The words FIVE PLUS TWO were printed because they were placed between the quotation marks, while the 7 was calculated because the 5 + 2 was placed OUTSIDE the quotation marks.

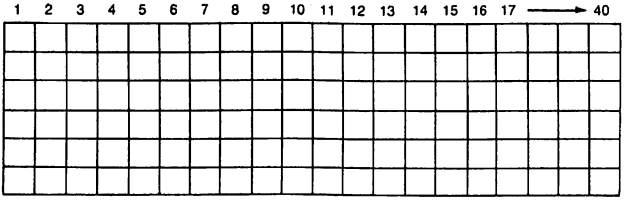
Remember those very important quotation marks:

PRINT "5 + 2" tells the computer to print whatever is between the quotes (5 + 2), but PRINT 5 + 2 tells the computer to CALCULATE 5 + 2 (the result is 7).

EXTRA INFORMATION

(A) MORE ABOUT FORMATTING THE OUTPUT:

If you wondered why the COMMA always puts columns in the same place on your Apple's monitor, here's a bit of an explanation. Your Apple computer divides the monitor's screen into 40 vertical columns.



You can display up to 3 fields using the COMMA, beginning in the 1st, 17th, and 33rd vertical columns.

You'll learn more about screen formatting in Lesson 6.

(B) USING YOUR COMPUTER AS A CALCULATOR:

The rules for using your Apple as a calculator sometimes are complicated. For example, suppose you typed:

```
PRINT 3 + 6 * 2 - 4
```

What do you think the computer would calculate as an answer?

Care to guess?

Actually, the computer follows certain mathematical rules whenever more than one type of arithmetic symbol appears.

RULES FOR PERFORMING ARITHMETIC OPERATIONS

1. All operations in parentheses are done first.
2. Exponentiation is done next.
3. Either multiplication or division is performed next.
4. Either addition or subtraction is performed next.
5. If two functions are equal in priority (for example, addition and subtraction) then your computer performs the math from left to right.

Let's follow these rules for the example:

```
PRINT 3 + 6 * 2 - 4
```

1. RULE #1: There are no parentheses
2. RULE #2: There are no exponents
3. RULE #3: The multiplication, 6*2 is performed (12)
4. Rule #4: The addition and subtraction are performed (3+12-4=11)

So your computer displays the answer, 11. (Did you get it before?)

Here's one more example:

```
PRINT 16 - (2+2) * 3 + 10  3
```

1. Rule #1: The operation in parentheses is done first:
   (2+2)=4
   So now you have 16 - 4 * 3 + 10 3

2. Rule #2: Exponentiation is carried out next:
   10 3=1000
   So now you have 16 - 4 * 3 + 1000

3. Rule #3: The multiplication (4*3) evaluates to (12).
   So now you have 16 - 12 + 1000

4. Rule #4: Addition and subtraction are performed:
   16-12=4
   4 + 1000=1004
   which is our final answer.

(C) THE TRUTH ABOUT QUOTATION MARKS:

We originally told you that quotation marks are needed both at the beginning and the end of a literal. In fact, Applesoft BASIC allows you to use a shortcut and disregard the closing quotes. Thus the command:

```
PRINT "THE BUS IS YELLOW
```

would give us the same output as the command:

```
PRINT "THE BUS IS YELLOW"
```

For the purposes of this program, though, we will require you to use quotation marks at the beginning and the end of all literals, since the use of this shortcut can lead to problems later on.

(D) A SHORTCUT TO USING PRINT:

The computer recognizes the question mark (?) as a PRINT command. Thus the statement:

```
? "THE BUS IS YELLOW"
```

has the same result as the statement:

```
PRINT "THE BUS IS YELLOW"
```

However, for the purposes of this program, the PRINT statement only will be recognized as the PRINT command.

A CHANCE TO REVIEW AND TRY OUT YOUR KNOWLEDGE

CHAPTER 2

PRACTICE AT THE COMPUTER:

For each of the following commands, write down what you expect the computer's response to be. Then try them out on your computer.

```
 1.  PRINT "HOW ARE YOU?"
 2.  PRINT "I WANT TO LEARN PROGRAMMING."
 3.  PRINT "YEAR", "CARS SOLD"
 4.  PRINT "AVERAGE"; "GRADE"
 5.  PRINT "7-3"
 6.  PRINT 7-3
 7.  PRINT 12*12
 8.  PRINT 8+2*2-2
 9.  PRINT "8 DIVIDED BY 2 IS "; 8/2
10.  PRINT "COUNTRY", "POPULATION", "INDUSTRY"
11.  PRINT 2*(3-1)+4/2
12.  PRINT 12/(7-4)+3*7^2
```

TRUE-FALSE

1. The PRINT statement is used to print any message, words or characters..
2. A semi-colon is used to create columns.
3. If you PRINT "WEL"; "COME" the word WELCOME will be displayed.
4. The slash (/) is the symbol that tells the computer to perform the arithmetic function of division.
5. If you type the command: PRINT "3+6", the computer will respond with 9.
6. During the arithmetic process the computer divides before it subtracts.

COMPLETION

1. The function of a computer that lets it print any message is called
2. The _____ is used to indicate that no space is to be left between words or characters when printed.
3. If you type in the command: PRINT "ONE"; "TWO"; "THREE" your computer will respond:
4. The _____ causes output to be printed in columns.
5. To multiply 25 times 5 and print the result, you type the command

ANSWERS

CHAPTER 2

AT THE COMPUTER:

1. HOW ARE YOU
2. I WANT TO LEARN PROGRAMMING
3. YEAR CARS SOLD
4. AVERAGEGRADE
5. 7-3
6. 4
7. 144
8. 10
9. 8 DIVIDED BY 2 IS 4
10. COUNTRY POPULATION INDUSTRY
11. 6
12. 151 (This expression evaluates to 12/3 + 3*49, which evaluates to 4 + 147).

TRUE-FALSE:

1. TRUE.
2. FALSE. A comma is used
3. TRUE, if you press the RETURN key.
4. TRUE.
5. FALSE. It will display: 3+6.
6. TRUE.

COMPLETION

1. Printing a literal
2. Semi-colon
3. ONETWOTHREE
4. Comma
5. PRINT 25*5

LESSON 3
ALL ABOUT STRINGS AND CONSTANTS

WHAT HAVE YOU LEARNED?

Lesson 3 was designed to teach you the difference between literals- called string constants- and numeric variables. You learned that the Apple handles strings differently from the way it handles numerics. You also learned how to assign strings and numerics to variables that the Apple can correctly recognize.

LET'S TAKE A CLOSER LOOK:

(1) String Constants:

Whenever we want to print a word in a program, we must enclose it in quotes. You may recall that the computer term for words within a program is "string constant."

Some examples of string constants are:

```
"HELLO"
"I LOVE YOU"
"ROBIN HOOD"
"FOURTH OF JULY"
```

The command:

```
PRINT "ROBIN HOOD"
```

will result in the following:

```
ROBIN HOOD
```

The computer recognizes the first quotation mark as the beginning of the string and the next quotation mark as the end of the string. It executes the PRINT statement for everything between the quotation marks.

Let's look at some other examples:

If You Type:	The Computer Will Print:
PRINT "I LOVE YOU"	I LOVE YOU
PRINT "E.T."	E.T.
PRINT "PHONE HOME"	PHONE HOME

You saw that the way in which the Apple recognizes the beginning and the end of a string may cause a problem if you attempt to use quotation marks within a string.

The following statement will confuse the computer and will cause a SYNTAX ERROR MESSAGE:

```
PRINT "I'VE SEEN "E.T." 3 TIMES, AND I CAN'T WAIT TO
SEE IT AGAIN."
```

Since the computer interprets everything between the first two quotation marks (I'VE SEEN) as the string to be printed, it will execute the PRINT statement for that phrase. After it does so, it has no instructions about what to do with the rest of the statement.

This can be corrected by using the apostrophe as a single quotation mark within the string:

```
PRINT "I'VE SEEN 'E.T' 3 TIMES, AND I CAN'T WAIT TO
SEE IT AGAIN."
```

which will result in the printing of the following:

```
I'VE SEEN 'E.T.' 3 TIMES, AND I CAN'T WAIT TO SEE IT AGAIN.
```

(2) The Differences Between String and Numeric Handling:

The number "12" enclosed in quotation marks is a string constant.

Thus the command:

```
PRINT "12"
```

will result in the following:

```
12
```

It may be difficult at this point to see the difference between strings and numerics, but remember, within the Apple they are handled in different ways.

A NUMERIC IS GIVEN A VALUE

It can be added to, subracted from, and multiplied or divided by another number.

ALL STRINGS HAVE A VALUE OF ZERO.

Thus, "12", "10", and "2578" all have a numeric value of zero because they are enclosed in quotation marks and are therefore considered by the computer to be strings.

If we choose to add two strings:

```
PRINT "10" + "5"
```

you learned that the computer will print them side by side:

```
105
```

The term that we use for this process of attaching strings by adding them together is CONCATENATION.

Let's look at another example of concatenation:

```
PRINT "I LOVE" + "MY" + "APPLE"
```

The result of this concatenation is

```
I LOVEMYAPPLE
```

(3) Variables:

Often when we are working with the Apple we want to save information in the computer's memory. (Remember that the computer has an area of memory called RAM which serves this very purpose.)

For instance, if you were doing the arithmetic for a shopping list, you might want the computer to "remember" that oranges are 89 cents per dozen or that apples are 39 cents per pound.

If you recall, Applesoft BASIC lets us store numbers and strings in variables.

A variable is simply a name given to a number.

Instead of referring to the number .39 (the price of apples), we can give it the name A. Then, if we later need to use the price of apples in our calculations, we can call for the variable A.

The BASIC statement LET allows us to give names to numbers. When we do this, we are making an assignment.

Let's make some assignments:

Assignment	Result
Let A = .39	The number .39 is stored in variable A
Let O = .89	The number .89 is stored in variable O
Let D = 12	The number 12 is stored in variable D

Using the first assignment as an example, let's follow the computer step by step as it carries out our command:

(1) The LET statement causes the computer to reserve a space in its memory cells. Initially, this space has no name.

(2) The variable A is used by the computer to refer to that space, and the only way you can get information from that space is by calling for it by its variable name (for instance, using the command PRINT A).

(3) The equal sign tells the computer what information to put in memory space A. In this case, you told it to put .39 in that memory space.

What would happen if you typed in the following program:

```
LET X = 10
PRINT X + 15
```

As you learned, the LET statement causes the computer to store the number 10 in memory location X.

Then the PRINT statement instructs the computer to take the contents of memory location X, add 15 to it and print the result.

The result, of course, is

```
25
```

(4) Variable names:

So far, we have used a few single letters to name our variables, but you may recall that we can use many letter combinations as variable names. In fact, Applesoft BASIC will recognize almost any combination of letters and numbers as a variable.

Thus, all of the following are legal variable names:

J7, AB, RALPH, ILOVEYOUAUNTSUE, B, QB37

You can use any combination of numbers and letters up to 255 characters to name a variable, provided that you abide by the following restrictions:

1. The variable name must start with a letter;

2. The variable name must not have any reserved words embedded in it;

3. The variable name must not have any blanks embedded in it.

These restrictions should never be a problem. As you become more familiar with Applesoft BASIC, you will recognize the words that are reserved for BASIC commands. You already know that the words LET and PRINT are reserved words, and in future lessons you will learn the uses of such other reserved words as IF, END, RUN, NEW, LIST, and GOTO.

The following is not a legal variable:

JIFFY

because the reserved word IF is embedded in it. The following is also not a legal variable:

LETTER

because the word LET is a part of it. These variables are also not legal:

7J3, 5X, 3BRIAN, X 17

Why?

Because all legal variables must begin with a letter, and blanks are not permitted as part of the variable name.

Up to this point in Lesson 3, we have used variables to represent numbers only, but we can also use variables to name strings.

String variable names follow the same rules as numeric variables, except that a dollar sign ($) must follow the variable name. This enables the computer to know that the information held in memory is to be treated as a string and not as a numeric.

All of the following are examples of legal string variables:

```
A$      LE$     HERMAN$      L7615$
O$      J7$     CAR$         BABERUTH$
```

A program may have, for example, a variable A and a variable A$, and because one is numeric and one is a string, the computer will not confuse them.

The following program shows assignments using string variables:

```
LET A$ = "CHRISTOPHER"
LET B$ = "COLUMBUS"
PRINT A$ + B$
```

The result is:

```
CHRISTOPHERCOLUMBUS
```

Just one reminder on the use of variables:

Although Applesoft BASIC permits variables of up to 255 characters, the computer recognizes only the first two characters of a variable name. Thus:

CAR and CAN are the same variable.

The sequence

```
LET SALEPRICE = 10
PRINT SALARY
```

will cause 10 to be printed because the computer "sees" only the first two characters of the variable name, and therefore does not distinquish between the two variables.

What will this sequence print?

```
LET CAR$ = "CORVETTE"
LET CAN$ = "TIN CAN"
PRINT CAR$
```

If you said TIN CAN, you're right!

Let's see why:

The first line assigns the word "CORVETTE" to the variable CA$ (Remember, the computer reads only the first two characters of the variable name).

The second line changes the contents of memory location CA$ to "TIN CAN" since CAR$ is the same as CAN$ as far as the computer is concerned.

The PRINT statement simply prints the value in CA$. At the time that it is called, CA$ is holding "TIN CAN."

Think you've got it? Try a few questions.

EXTRA INFORMATION

The LET Statement:

When we introduced the LET statement, we indicated that a number or string could be assigned only to a variable using a statement beginning with the word LET.

Actually, this is not true on the Apple. The fact is that the computer does not need the word LET in order to recognize an assignment statement. Thus,

```
LET APPLE$ = "COMPUTER"
```

is the same as

```
APPLE$ = "COMPUTER
```

and

```
LET RENT = 250
```

is the same as

```
RENT = 250
```

COMPUTER EXERCISES

Correct the error in each of the following lines: (The answers are on page 29)

```
1.)  PRONT "I'LL HAVE A HOT DOG WITH MUSTARD"
2.)  PRINT HUMPHREY BOGART
3.)  PRINT RUDOLF NUREYEV"
4.)  LET MISPRINT = "BASSBALL"
5.)  LET A = "EXCELLENT"
6.)  LET B$ = 7
```

For each of the following, enter ALL lines of code. Can you guess what will be printed?

```
A)  LET NUMBER = 5
    PRINT NUMBER

B)  LET NAME$ = "JOHN"
    PRINT NAME$

C)  LET NAME$ = "JOHN"
    LET NUMBER = 14
    PRINT NAME$; " IS "; NUMBER; " YEARS OLD"

D)  LET ANSWER$ = "TRUE"
    LET ANSWER$ = "FALSE"
    PRINT ANSWER$

E)  LET CALC1 = 6
    LET CALC2 = 8
    PRINT CALC1 * CALC2

F)  LET AN$ = "HELLO "
    LET AM$ = "JIM"
    PRINT AN$ + AM$

G)  LET HOURS = 5
    LET RATE = 4
    LET MSG$ = " HOURS YIELDS "
    PRINT HOURS; MSG$; "$"; HOURS * RATE
```

A CHANCE TO REVIEW AND TRY OUT YOUR KNOWLEDGE

TRUE/FALSE:

1. The statement:
   LET A = 5
   is treated by the Apple in the same way as the statement:
   A = 5

2. The computer term for words within a program is "numeric variable."

3. In order to use quotation marks within a string it is necessary to give special instructions to the computer.

4. The number "347" is a numeric.

5. The number 456 is a numeric

6. The command PRINT "50" + "35" will result in the printing of the string 5035.

7. Attaching strings by adding them together is known as numeric variability.

8. A name given to a number is called a variable.

9. LET V = 5 is an example of an assignment statement.

10. The computer will interpret the variables MAX and MANCHURIA as if they are the same variable.

MULTIPLE CHOICE QUESTIONS

1. Which is a correct PRINT command?
   A) PRINT "MAID MARIAN"
   B) PRINT 'LITTLE JOHN'
   C) PRINT FRIAR TUCK

2. What will be printed for the following Print Statement:
   PRINT "BABE RUTH IS KNOWN AS 'THE SULTAN OF SWAT'"
   A) BABE RUTH IS KNOWN AS
   B) 'THE SULTAN OF SWAT'
   C) "BABE RUTH IS KNOWN AS 'THE SULTAN OF SWAT'"
   D) BABE RUTH IS KNOWN AS 'THE SULTAN OF SWAT'

3. Which is a legal command:

A) PRINT "10" + 5 B) PRINT "10" + "5"

4. Which of the following is a legal numeric variable:
A) A$ B) B C) 3C D) IF

5. Which of the following is a legal string variable:
A) ALICE B) FRANK C) JOHN$ D) 7EDWARD

6. Which of the following is not a legal variable:
A) J1$ B) 1J$ C) JOHN D) OHJN

7. Which of the following is a legal LET statement:
A) LET Z = 47 B) LET 47 = Z C) LET X = "TRUMAN"

8. Which of the following is not a legal LET statement:
A) LET X = 39.4 B) LETXXX = 371 C) LETX$ = 470

9. Which of the following is a reserved word:
A) PRINT B) HEART C) CAN D) CAR

ANSWERS CHAPTER 3

CORRECT THE LINES:

1. Print is spelled incorrectly, so the computer will not recognize the instruction.
2. Quotation marks are missing.
3. Opening quotation mark is missing.
4. MISPRINT is an illegal variable name because the reserved word PRINT is embedded within it.
5. A string variable (such as A$) is needed.
6. A numeric variable must be used with numeric data.

TRUE/FALSE:

1) T
2) F-string constant
3) F-use apostrophes in place of quotation marks.
4) F-since it is enclosed in quotation marks, it is a string.
5) T
6) T-This is an example of concatenation.
7) F-This is the definition of concatenation.
8) T
9) T
10) T

MULTIPLE CHOICE:

1) A All strings must begin with double quotation marks.
2) D The string within the quotation marks is printed exactly as written, but without the quotation marks.
3) B Strings and numerics cannot be mixed in arithmetic.
4) B Choice A is a string variable, C and D are illegal variables.
5) C All string variables are followed by dollar signs.
6) B Variables must begin with a letter.
7) A The assignment variable must be first; C is an attempt to assign a string to a numeric variable.
8) C X$ is a string variable; strings must be in quotation marks.
9) A

LESSON 4
AN INTRODUCTION TO PROGRAMMING

WHAT HAVE YOU LEARNED?

You've completed lesson 4 on the computer and so congratulations are in order, because now you're a programmer!

In this lesson, you learned the meaning of the word "program" and the importance of line numbers in a program. You used the command RUN to get the computer to execute your program and the command LIST to view the instructions line by line. You also discovered that you could add lines anywhere within your program by assigning them new line numbers, and that you could start from scratch using the command NEW.

LET'S TAKE A CLOSER LOOK:

(1) What's a program?

In a sense, you've already been programming, but in previous lessons you were telling the computer to process only one statement at a time.

A computer program is a set of statements which work together to perform a specific task.

Instead of entering one statement at a time, Lesson 4 showed you how to write many statements at once and then process those statements as a group.

The following is an example of a program:

```
100 LET X = 5
120 LET Y = 2
140 PRINT X + Y
160 END
```

Line 100 assigns the number 5 to variable X
Line 120 assigns the number 2 to variable Y
Line 140 instructs the computer to add the contents of variable X to the contents of memory space Y and print the result.
Line 160 tells the computer that there is nothing more to do.

When you type in a program like the one above, the computer does not execute any of the lines until told to do so. As you discovered in Lesson 4, the command RUN- without a line number- is the instruction that will cause the computer to execute your program.

(2) Line Numbers:

Line numbers tell the computer the order in which it should execute your instructions. The computer will always execute in order from the lowest to the highest line number, but how do you decide which line numbers to use?

First, the Apple places a few restrictions on line numbering:

(1) The line number must be a positive integer.
(2) The line number cannot be greater than 63999.

The following are legal line numbers:

```
    1
   10
 3002
   19
 1445
60753
20000
```

(Notice that no commas are allowed in line numbering!)

The following are illegal line numbers:

64.75 83.2 -73 -10 (These are not positive integers)
65791 (This number is higher than the limit of 63999)

Second, there are certain considerations that you should take into account when choosing line numbers for your program.

Let's look at our previous program numbered in two ways- we'll call these ways "right" and "wrong."

RIGHT

```
100 Let X = 5
120 Let Y = 2
140 PRINT X + Y
160 END
```

WRONG

```
1  Let X = 5
2  Let Y = 2
3  PRINT X + Y
4  END
```

In the "right" program the numbers are not consecutive. We have purposely left room between them so that we can add additional lines to our program. There is, in fact, room to add 19 additional lines between each two statements.

In the "wrong" program there is no place for additional lines.

Now, if we wanted to add the line:

PRINT "THE ANSWER IS"

we could easily place it betwen lines 120 and 140 in the "right" program, but we have no room for it in the "wrong" program.

As a rule, you should always leave space to add new lines, but the amount of space is entirely a matter of your preference and your estimate of the needs of your program.

(3) Programming Commands:

In Lesson 4 you were introduced to three commands that you will use often as a programmer:

(A) NEW:

Every time you start a new program, you should enter the command NEW. This command tells the computer to clear its memory of any variables that may have been left from earlier programs. It also tells the computer to erase any program that is in its memory.

When NEW is entered, the computer initializes all numeric variables to zero and all string variables to blank spaces.

If you are at a computer now, you can see this for yourself by following along step-by-step:

(a) Put your start-up disk in the drive and turn on your system.

(b) Enter the command NEW.

(c)

Now, enter the command PRINT A.

Since all numeric memory cells have been set to zero, the computer will respond by displaying a zero.

(d)

Now assign the value of 8 to A by entering the following line:

LET A = 8

(e) Enter the following line:

```
PRINT A
```

The computer should respond with:

```
8
```

since you assigned the value of 8 to variable A.

(f) Again, type NEW.

You have now cleared the memory.

(g) Finally, type:

```
PRINT A
```

The computer will respond with a zero. Variable A no longer has the value of 8 because NEW told the computer to clear all its variables.

(B) RUN:

Having learned how to use the NEW command to begin a program, let's see if you recall how to get the computer to execute the program once you've typed it in.

After NEW is entered and all of the lines of your program are entered (along with their line numbers), the command RUN tells the computer to execute each line of the program.

Try entering the previous program using the additional commands NEW and RUN.

(Notice that neither has a line number!)

```
NEW
100 LET X = 5
120 LET Y = 2
140 PRINT X + Y
160 END
RUN
```

The computer will print the number 7.

Note that at this point the program is still in memory. We have not erased it, and we can execute it as often as we like by typing RUN each time we want to use it. To demonstrate this, you may enter the command LIST.

(C) LIST:

You were introduced to the LIST command in this lesson as a way of displaying the latest version of your program.

Any time you change your program, LIST will display the most recent copy of your program.

Let's add a new line to the program we just ran:

```
125 PRINT "THE ANSWER IS "
```

(Remember to press RETURN after typing each line!)

Now enter LIST. You should see the following:

```
100 LET X = 5
120 LET Y = 2
125 PRINT "THE ANSWER IS "
140 PRINT X + Y
160 END
```

LIST displayed the current version of your program in memory, placing the new line in its correct numerical place within the program.

What would happen if you typed the following two commands:

```
NEW
LIST
```

Since NEW clears the memory and LIST displays the memory, nothing would be printed!

EXTRA INFORMATION

1) THE LIST COMMAND:

The LIST command by itself always lists a program from the first line to the last line. This can be inconvenient if you are working with a lengthy program. Instead of waiting for hundreds of lines to scroll by, you may use several other forms of LIST to bypass the lines that you don't want to see. Take a look at the examples below:

A. LIST 100-200
Lists line 100 to line 200.

B. LIST 100,200
The comma and dash can be used interchangeably. This line will do the same thing as the previous line.

C. LIST 100-
This command will list from line 100 until the last line of the program.

D. LIST 100
Lists only line 100- no other lines.

E. LIST-
This is the same as LIST without the dash. Without a number before the dash, the computer assumes zero, so the program will be listed starting at line 0 or at the lowest numbered line.

There are also other tricks that you can use with the LIST command to help you view the program more easily:

F. If you press the CONTROL KEY and the letter 'S' together while the program is listing, the listing will pause until you press another key (any other key).

G. If you press the CONTROL KEY and the letter 'C' while the program is listing or while the computer is doing anything else, the listing will stop, and the computer will be ready to do something else.

2) THE REM STATEMENT:

Now that you are beginning to write your own programs, you should know about a very important feature of BASIC, called the REM statement.

REMs (short for remarks) are used to document a program. When the computer sees a REM, it skips over it to the next line. Since it is not executable, it has no effect on the RUNning of a program. However, it can be seen when the program is LISTed. Look at the following example:

```
10  REM AUTHOR - JOHN DOE
20  REM THIS PROGRAM ADDS TWO NUMBERS
30  PRINT 5 + 6
```

When this program is executed, only the number 11 will be printed. However, when the program is listed, all 3 lines will be displayed.

It is good programming practice to use REM statements throughout your program. As your programs become more complicated, you will find that REMs make it easier for you or for another person looking at your program to understand what your variables represent and what your program is doing at important points.

3) THE TRUTH ABOUT END:

We told you that the END statement is needed as the last line of all BASIC programs. The truth is that the Apple will halt execution of your program even without the END statement when it reaches the last line of your program, so that END really isn't needed at all! However, you should acquire the habit of using the END statement at an early stage of your programming career so that you are using it routinely when you reach the stage at which its use may be important.

COMPUTER EXERCISES

For each number below, enter all of the lines. Before each exercise, try to guess what will happen.

(1)
```
NEW
10 PRINT "EXERCISE1"
RUN
LIST
```

(2)
```
NEW
460 X = 3
480 Y = 2
500 PRINT X/Y
LIST
RUN
```

(3)
```
NEW
100 PRINT "THE ANSWER IS "
150 PRINT 6 + 6
LIST
RUN
90 PRINT "WHAT IS 6+6?"
LIST
RUN
```

(4)
```
RUN
LIST
NEW
RUN
LIST
```

(5)
```
NEW
2000 HOURS = 40
2010 RATE = 4.20
2020 PRINT "YOUR WEEKLY SALARY
  IS ";RATE*HOURS
RUN
LIST
```

(6)
```
NEW
60 HOURS = 30
75 RATE = 5
LIST
```

(7)
```
80 PRINT "HOURS", "RATE"
85 PRINT HOURS, RATE
LIST
RUN
```

(8)
```
75 RATE = 5.65
RUN
```

(9)
```
60 HOURS = 20
LIST

 RUN
```

(10)
```
NEW
100 A = 10
110 B = 20
120 C = 50
140 PRINT "THE
    AVERAGE IS ";
    (A + B + C)/3
```

ADDITIONAL PRACTICE PROGRAMS:

1. Write a program to:

   A. Multiply 6 by 3
   B. Add 2 to 164
   C. Add a line between A and B to divide 8 by 5

2. Write a program that:

   1. Adds 8 + 8
   2. Multiplies 8 times 8
   3. Subtracts 8 from 8
   4. Divides 8 by 8

When you have completed and run the program, change all the eights to sixes, then add a line between multiplication and subtraction that prints the following message: MULTIPLICATION OF A NUMBER BY ITSELF IS KNOWN AS RAISING IT TO THE SECOND POWER.

SUGGESTED ANSWERS

ADDITIONAL PRACTICE PROGRAMS

1. A) 10 PRINT 6 * 3
   B) 20 PRINT 2 + 164
   C) 15 PRINT 8/5

2.
```
10  PRINT "8 + 8 = "; 8 + 8
20  PRINT "8 X 8 = "; 8 * 8
30  PRINT "8 - 8 = "; 8 - 8
40  PRINT "8 / 8 = "; 8 / 8
25  PRINT "MULTIPLICATION OF A NUMBER BY ITSELF RAISES
IT TO THE SECOND POWER."
```

LESSON 5
A FIRST LOOK AT THE INPUT STATEMEN

WHAT HAVE YOU LEARNED?

Lesson 5 took you a few steps further down the road to serious programming.

You learned how to use the HOME command to clear the screen before running a program and some of the many ways that the INPUT statement can be used to make a program more interesting and/or user-friendly.

You also learned how to insert, delete and replace lines within your programs and how to SAVE programs on a disk and then LOAD them again when needed.

LET'S TAKE A CLOSER LOOK:

(1) The HOME statement:

The HOME command serves two purposes: first, it clears the screen; second, it tells the computer to move the cursor to the upper left-hand corner of the screen.
It is a useful tool for formatting your output.

Let's look at an example:

```
10  HOME
20  PRINT "HELLO"
30  PRINT "MY NAME IS JOHN"
40  END
```

Line 10 of this program instructs the computer to clear the screen and move the cursor to the upper lefthand corner.

Lines 20 and 30 tell the computer to PRINT a message. Since the cursor has been moved to the upper lefthand corner, that is where printing will begin.

The output will appear as follows:

```
HELLO
MY NAME IS JOHN
```

Remember: HOME only clears the screen.It does not erase any of the contents of the Apple's memory.

(2) The INPUT statement:

The INPUT statement is used to get information from the person using the program. You saw that it must always be followed by a variable (which assigns memory space within the computer where the information you are asking for will be held.)

```
For example:
100 PRINT "HOW OLD ARE YOU"
110 INPUT A
120 PRINT "YOU ARE "; A; " YEARS OLD."
```

When this program is executed, line 110 causes a question mark to be displayed. When this question mark appears, the user is expected to enter his or her age. The computer waits as long as necessary for a response, and when it receives one, it stores the response in the memory space reserved for variable A. (Notice that input for a numeric variable must be numeric. If a string is entered where a number is expected, the computer will tell you to redo from the start. If a number is entered where a string is expected, the number will be treated as a string).

Let's take a step-by-step look at this program as it runs. The items in parentheses are our comments:

```
HOW OLD ARE YOU          (This is the execution of the PRINT
                          statement on line 100)

?17                      (Question mark appears as a result of
                          INPUT statement on line 110.  User
                          responds by typing 17, which is
                          stored in variable A.)

YOU ARE 17 YEARS OLD.    (This is the execution of the PRINT
                          statement on line 120.  Recall that
                          a semi-colon causes the cursor to
                          stop one space after the last
                          printed character.)
```

(Try the above example for yourself, or vary it to ask for other pieces of information, but remember, the variable that you use for INPUT must be of numeric type if you are expecting a number, and string type if you are not!)

INPUT may be used in conjunction with any legal variable. Therefore, the statements below are all legal:

```
 50 INPUT Q
100 INPUT A$
120 INPUT NAME$
140 INPUT RENT
```

The INPUT statement is a powerful programming tool because many of your programs will ask the user for information.

When you have access to a computer, you should enter and run the following programs:

```
PROGRAM A:
10 PRINT "ENTER A NUMBER"
20 INPUT N
30 PRINT "YOUR NUMBER MULTIPLIED BY 18 IS "; 18 * N
40 END

PROGRAM B:
10 PRINT "WHAT TOWN DO YOU LIVE IN"
20 INPUT T$
30 PRINT "YOU LIVE IN "; T$
40 END
```

There are other ways that an INPUT statement can be structured. For example, instead of typing:

```
110  PRINT "WHAT IS YOUR NAME?"
120  INPUT NA$
```

You could type:

```
110 INPUT "WHAT IS YOUR NAME?";NA$
```

The Apple permits us to combine the INPUT and PRINT statements in a single INPUT statement, but when we do so, it does not generate its own question mark, so that we had to provide our own in this case.

When run, this line produces the following output:

```
WHAT IS YOUR NAME?
```

The computer waits at this point for a string to assign to NA$. This information must be provided by the user, who will press RETURN after doing so in order to tell the computer that it can store the information and move on.

One more example:

```
760 INPUT "YOUR MESSAGE HERE"; Z$
```

A word of caution:

DON'T FORGET QUOTATION MARKS AROUND THE LITERAL, AND DON'T FORGET THE SEMI-COLON THAT SEPARATES THE VARIABLE NAME FROM THE LITERAL.

(3) Program editing:

In Lesson 4 you learned how to insert a line in your program. Providing that you left room between your line numbers, all you had to do was to enter the new line with the appropriate line number.

Lesson 5 took you further, though. You learned how to change and delete lines as well.

Take the following program:

```
10  PRINT "WHAT IS YOUR NIME"
20  INPUT NAME$
```

If you wanted to change line 10 to make it read:

```
10  PRINT "WHAT IS YOUR NAME"
```

You could do so simply by typing the above line and pressing RETURN. Using the same line number will cause the computer to replace the original line with the later version.

Suppose you wanted to delete a line from your program.

In Lesson 5 you saw that deletion is very easy. To delet a line, all you need do is enter the line number of the line t be deleted and then press the RETURN key.

For example:

```
210  PRINT "APPLE
230  PRINT "ORANGE"
250  PRINT "COMPUTER"
```

To delete line 230 in the above example, you would enter the number 230 and then press RETURN.

If you were then to LIST the program, you would see the following:

```
210  PRINT "APPLE"
250  PRINT "COMPUTER"
```

To delete more than one line at a time, there is a separate delete command:

DEL (first line number), (last line number)

If you type DEL followed by the first line to be deleted, then a comma, then the last line to be deleted the computer will erase from its memory all of the lines between the first and last including both the first and the last line.

For example:

DEL 400,1000

deletes all the lines from 400 to 1000 including the starting line (400) and the ending line (1000).

PROGRAM EDITING REVIEW

TO INSERT A LINE. . .

Enter the new line using a line number between the two lines where you want it inserted.

TO REPLACE A LINE. . .

Enter the new line using the same line number as the one you want to replace.

TO DELETE A LINE. . .

Enter the line number of the line you want deleted and press RETURN.

(4) Using disks:

A. SAVING PROGRAMS

The disk drive on your Apple allows you to save programs and text permanently, so that if you write a program today that you expect to use again, you can save it and recall it to the computer rather than typing it over each time you want to use it.

Let's see how this is done.

First, put your work disk in the drive and enter the program below: (If you have not yet prepared a work disk, Appendix B at the back of this manual explains how to do so.)

```
10  PRINT "MY FIRST PROGRAM"
20  END
```

Next, enter the following line:

```
SAVE PROG1
```

When you do this, you should hear the disk drive whirr and also see the red light on the front of your drive come on for a short time.

To verify that your program is on the disk, enter the command CATALOG. As you learned earlier, CATALOG is the comman that enables you to see the disk's directory (which is similar to a book's table of contents).

You should see the name of your program displayed along with some other information. (For details, see the Extra Information pages at the end of this lesson.)

Now, enter the command NEW. Recall that NEW clears the computer's memory. Next, type LIST. You should see nothing displayed since you have removed your PROG1 from the computer memory.

Suppose now that you want your old program back. You can retrieve it with the LOAD command:

```
LOAD PROG1
```

LOAD tells the computer to find the specified program on the disk and transfer a copy of it to the computer's memory. The program is copied from the disk so that it now exists in tw places: on the disk and in the computer's memory. To prove this, type CATALOG. You will see that the program is still in the disk directory.

B. INITIALIZING A DISK

When dealing with a new disk, you must format (or initialize) it so that the computer can locate programs and create a directory. The command for formatting a new disk is INIT. (short for initialize). You have previously used INIT when you set up your work disk.

Generally, when you introduce a new disk, you would first start the system with your DOS disk, then enter lines similar to the following:

```
NEW
10 PRINT "WRITTEN JAN 1, 1983
20 END
INIT HELLO
```

The above steps cause a program named "HELLO" to be the first program on your disk. Many programmers consider it a good idea to use the same name to initialize all their disks.

Once the disk is initialized, the "HELLO" program will automatically run whenever the disk is BOOTED (by turning on the computer or by entering the command PR#6 on an APPLE II Plus or by simultaneously pressing CONTROL, RESET, and OPEN APPLE on an APPLE IIE).

Caution!! INIT will erase any programs that might previously have been stored on a disk, so be sure to use INIT only on new disks.

EXTRA INFORMATION

1. Reading the Disk Directory:

When you read the disk directory, you should see the name of your program displayed, along with some other useful information. The letter all the way on the left of the screen tells you what kind of program is stored on the disk.

"A" is for an Applesoft BASIC program. "I" is for an Integer BASIC program, "B" is for a binary program and "T" is for a text file. PROG1 should have an "A" to the left of it, specifying an Applesoft program. Also to the left of PROG1 is a number. This number specifies the number of sectors your program takes up on the disk. In other words, the number tells you how large your program is.

2. Deleting Disk Files:

Disk files can be deleted just as they can be saved. The command:

```
DELETE PROG1
```

will erase the program from the disk, thereby leaving room on the disk for a replacement file.

QUIZ

I. MATCHING:

.....CATALOG	A. Retrieve a program from disk
.....INPUT	B. Enter only a line number
.....HOME	C. Enter the new line with the line number of the one to be replaced
.....INSERTING	D. Obtain information from the user of the program
.....DELETING	E. Clear the screen and move the cursor to the upper left-hand corner of the screen
.....SAVE	F. Enter the new line with a line number between the lines you want to replace
.....LOAD	G. Format a disk
.....INIT	H. Store a program on a disk
	I. Find out you what programs are on the disk

II. CORRECT THE ERROR IN EACH LINE OF THIS PROGRAM

```
10  INPUT "WHAT IS YOUR NAME?"; NAME
20  INPUT "WHAT IS YOUR BATTING AVERAGE?" AV
30  PRINT NAME", "BATTING AVERAGE"
40  PRINT NAME$, PERCENT
50  RUN
```

III. TYPE IN THESE PROGRAMS AND PREDICT THEIR OUTPUT:

(1)
```
NEW
10  PRINT "HELLO"
20  PRINT "MY NAME IS ";
30  PRINT "GODZILLA"
40  PRINT "WHAT IS YOUR NAME"
50  INPUT N$
30  PRINT "KING KONG"
10
35  PRINT "AND I'M AWFULLY MEAN!"
```

(2)
```
NEW
100  INPUT "TYPE A NUMBER BETWEEN 1 AND 5"; NUMBER
125  PRINT "IF IT'S HIGHER THAN 4 I KNOW WHAT IT IS."
150  PRINT "YOUR NUMBER IS"; NUMBER
125  PRINT "I BET I CAN GUESS YOUR NUMBER."
160  PRINT "SEE!  THE COMPUTER NEVER MAKES A MISTAKE!"
```

ANSWERS: CHAPTER 5

I. MATCHING

CATALOG-I; INPUT-D; HOME-E; INSERTING-F; DELETING-B; SAVE-H; LOAD-A; INIT-G

II. CORRECT THE ERROR:

LINE 10: NAME SHOULD BE NAME$ SINCE THE QUESTION CALLS FOR STRING INPUT

LINE 20: SEMICOLON IS MISSING BETWEEN LITERAL AND VARIABLE AV

LINE 30: BEGINNING QUOTE FOR NAME IS MISSING

LINE 40: PERCENT IS NOT THE VARIABLE NAME, AV IS

LINE 50: RUN SHOULD BE TYPED WITHOUT A LINE NUMBER

III. OUTPUT

(1)
```
MY NAME IS KING KONG
AND I'M AWFULLY MEAN
WHAT IS YOUR NAME
?
```

(2)
```
TYPE A NUMBER BETWEEN 1 AND 5
(USER TYPES 3)
I BET I CAN GUESS YOUR NUMBER.
YOUR NUMBER IS 3
SEE!  THE COMPUTER NEVER MAKES A MISTAKE!
```

ADDITIONAL PRACTICE PROGRAMS

1. Write a program to compute the area of a rectangle. You should ask the user for the length and width of the rectangle.

2. Write a program that accepts a cost and a sales tax. Print the total cost of the item purchased.

3. Write a program to convert miles to kilometers (1 mile=1.609 kilometer). Use the following test data:

a.) 3 b.) 14
c.) 37.5 d.) 0

4. Given two inputs-minutes and seconds, write a program to print the total time in seconds. Use the following numbers:

a.) 3 min 12 sec c.) 2 min 11 sec
b.) 39 sec d.) 6 min 5 sec

SUGGESTED ANSWERS

ADDITIONAL PRACTICE PROGRAMS

1.
```
NEW
10  PRINT"ENTER LENGTH OF RECTANGLE"
20  INPUT L
20  PRINT "ENTER WIDTH OF RECTANGLE"
30  INPUT W
40  PRINT"THE AREA IS ";L * W
```

2.
```
NEW
10  INPUT PRICE
20  INPUT TAXRTE
30  PRINT "PRICE: "; PRICE,"TAX RATE:" ; TAXRTE
40  LET TAX = TAXRTE*PRICE
50  LET COST = PRICE+TAX
60  PRINT "TOTAL TAX: "; TAX, "TOTAL COST: "; COST
70  END
```

3.
```
NEW
10  PRINT "MILES TO KILOMETERS"
20  PRINT "1 MILE = 1.609 KILOMETERS"
30  INPUT "MILES" "; MILES
40  PRINT "KILOMETERS: ";MILES * 1.609
50  END
```

4.
```
NEW
10  PRINT "TIME IN SECONDS"
20  INPUT "INPUT MINUTES"; MIN
30  INPUT "INPUT SECONDS"; SEC
40  PRINT "TOTAL SECONDS ="; MIN * 60 + SEC
50  END
```

LESSON 6
FORMATTING THE OUTPUT

WHAT HAVE YOU LEARNED?

Lesson 6 introduced you to some Applesoft BASIC commands that make it possible for you to format your output. You learned how and when to TAB, HTAB and VTAB so that your PRINT statements appear as you want them to on the screen.

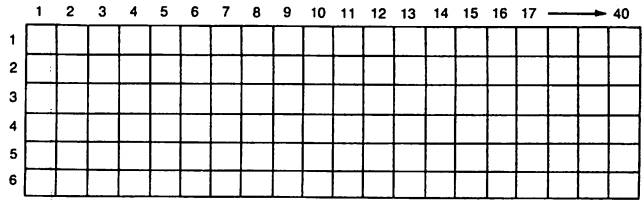
You also learned about printing modes. You can now print your output normally or have all of it or a portion of it appear in inverse or flashing mode.

With each lesson, the programming tools at your command make your programs more enjoyable, more interesting and more complex.

LET'S TAKE A CLOSER LOOK:

(1) Mapping The Sceen:

You saw that the Apple screen is divided into 24 rows of 40 columns. We can visualize the screen in the following way:



The Apple is capable of printing one character in each space of this grid. Three commands, TAB, HTAB and VTAB allow you to begin printing output at any point on the grid.

(2) The TAB Command:

Since Lesson 2 you have had some capability of controlling cursor movement and output print positions. You could choose to use a semi-colon or a comma in a print statement to format your output, but that gives you only a limited choice of how the output should look.

In this lesson, you learned how to use the TAB command to begin printing in a particular column.

For instance, if you want to print the word "HELLO" starting in column 18, you can use the following command:

```
PRINT TAB (18) "HELLO"
```

The number in parentheses tells the computer in which column the printing should begin. The number must be positive, but it doesn't have to be an integer.

The following are legal commands:

```
PRINT TAB (12) "MY NAME IS ADAM"
PRINT TAB (18.5) "HELLO"
PRINT TAB (39) "GOODBYE"
```

The following is not a legal command:

```
PRINT TAB (-17) "HELLO"
```

because the number in parentheses is not a positive number.

Let's see how the Apple deals with a TAB statement that does not use an integer:

```
PRINT TAB (18.5) "HELLO"
```

This statement has the same effect as:

```
PRINT TAB (18) "HELLO"
```

because real numbers are truncated (that is, everything after the decimal is chopped off) in TAB statements.

What will be the result of this statement?

```
PRINT TAB (8.39) "GOODBYE"
```

If you said that the (8.39) would be treated as an (8) you were right! Printing will begin in the 8th column.

TAB statements are not limited to constants. Expressions can be used as well:

```
PRINT TAB (12 * X) "HELLO"
```

Since X is a variable, the computer will multiply 12 by whatever is in variable X. If, for example, X were equal to 3, where would printing begin?

Since 12 times 3 is 36, the above statement would be the same as:

```
PRINT TAB (36) "HELLO"
```

(3) The HTAB Statement:

You learned in this lesson that HTAB is similar to TAB in its effect, but that HTAB can be used by itself.

```
10  HTAB(6)
20  PRINT "HELLO"
```

This program directs the computer to display the word "HELLO" in column 6 of the current line and has the same effect as:

```
10  PRINT TAB(6) "HELLO"
```

Often, programmers find it easier to read formatting statements that are placed on separate lines, so that changing these statements in situations in which print positions are important may be somewhat more convenient using HTAB than TAB. The difference isn't worth too much of your conern at this point, though.

(4) The VTAB statement:

As you saw at the computer, VTAB is an Applesoft statement that allows you to select the row at which printing will take place.

```
50  VTAB(8)
60  PRINT "CAR"
```

The example above instructs the computer to display the word "CAR" in the eighth row of the screen. A legal VTAB statement must have a real number between 1 and 24 in the parentheses.

The following statements are legal:

```
VTAB(24)
VTAB(1)
VTAB(3)
VTAB(8.5) (evaluates to VTAB(8))
VTAB(2.1) (evaluates to VTAB(2))
VTAB(6.99) (evaluates to VTAB(6))
```

since all reference numbers between 1 and 24.

The following statements are not legal:

```
VTAB(0)  (out of range-too low)
VTAB(-3) (out of range-too low)
VTAB(35) (out of range-too high)
```

(5) Printing Modes:

Besides formatting your output, you discovered in Lesson 6 that Applesoft Basic has some "fancy" features.

Three statements, INVERSE, NORMAL and FLASH allow you to change the printing mode as desired.

A. INVERSE. . .

Normally, as you have seen, the Apple displays white letters on a black screen. You can, however, direct the Apple to display black letters on a white background as the following example demonstrates:

```
15  INVERSE
20  PRINT "WOW"
25  NORMAL
```

Line 15- the INVERSE command- tells the computer to change the mode of printing to black letters on a white background.

Line 20 is printed in inverse mode, then line 25 tells the computer to resume printing in a normal mode.

If line 25 were not part of the program, everything displayed on the screen after the INVERSE command would be in inverse mode.

B. FLASH. . .

FLASH directs the computer to make the output of the PRINT statement blink on and off. This has the effect of having the output look like a neon sign.

Some of the exercises at the end of this section will use the FLASH statement.

EXTRA INFORMATION

1. SPACING IN PROGRAM LINES:

Throughout this series of lessons, whenever we have given programming examples we have written our statements so that they would be as readable as possible.

For instance, the line

```
10  PRINT "APPLE"
```

has two blank spaces. Applesoft Basic actually ignores the spaces between statements, so that if you were to type this line as follows:

```
10PRINT"APPLE"
```

it would be just as acceptable to the computer. Spaces between words and numbers are for the convenience of humans, not computers. Therefore, even though they are not necessary, we strongly recommend that you continue to use them to enhance the readability of your programs.

2. WHAT'S IN A LINE?

Now seems a good time to mention another useful feature of Applesoft BASIC. This is the ability to put more than one programming statement on a single line.

This is done using a colon (:) to separate different instructions or statements:

```
10 VTAB(4): HTAB(6)
20 PRINT "HELLO"
```

When the Apple "sees" the colon, it "knows" that it has reached the end of a statement.

In fact, the above program could have been written as a single line:

```
10 VTAB(4):HTAB(6):PRINT "HELLO"
```

This single line instructs the computer to print "HELLO" beginning on the 4th line, fifth column of the screen.

3. A PARENTHETICAL REMARK:

Parentheses are not necessary for the column or line numbers used with VTAB and HTAB. The statements:

```
10  VTAB(5)
20  HTAB(6)
```

are the same as

```
10  VTAB 5
20  HTAB 6
```

For this program, though, we'll continue to use VTAB and HTAB in parentheses only.

4. MORE ABOUT TAB AND HTAB:

Even though the screen is only 40 columns wide, TAB and HTAB can be used with numbers as high as 255. When the Apple encounters a TAB or HTAB statement with a number greater than 40 it skips one row for every 40 columns and tabs over the balance.

So the BASIC line:

```
10 VTAB (5): HTAB (55)
```

is the same as the line:

```
10 VTAB (6): HTAB (15)
```

QUIZ

TRUE OR FALSE:

1.) .....The line: HTAB(7): PRINT "HELLO" has the same effect as the line: HTAB (7.6): PRINT "HELLO".

2.) .....VTAB is used to specify what row to print in,and HTAB is used to specify a column.

3.) .....VTAB (16.8) would cause any output to be displayed in row 17 of the screen.

4.) .....VTAB (5): PRINT TAB (100) "HELLO" is a legal BASIC line.

5.) ..... The statement NORMAL directs the computer to display black letters on a white screen.

COMPUTER EXERCISES

At the computer, enter and execute each of the following groups of lines.

(A)
```
NEW
10 VTAB(12):HTAB(7)
12 PRINT "WHERE AM I"
RUN
```

(B)
```
NEW
10 PRINT TAB(6)"HI THERE"
RUN
```

(C)
```
NEW
10 HTAB ((40-6)/2)
20 PRINT "CENTER"
RUN
```

(D)
```
15 INVERSE
LIST
RUN
NORMAL
```

(E)
```
15 FLASH
LIST
RUN
NORMAL
```

(F)
```
15 INVERSE
RUN
LIST
25 NORMAL
RUN
LIST
```

(G)
```
NEW
10 PRINT "ENTER ROW"
20 INPUT ROW
30 PRINT "ENTER COLUMN"
40 INPUT COLUMN
50 VTAB (ROW)
```

```
60 HTAB (COLUMN)
70 PRINT "WHERE?"
RUN
?5
?18
RUN
```

ANSWERS: TRUE/FALSE

(1) TRUE. The number in parentheses is truncated in TAB, HTAB, and VTAB statements.

(2) TRUE.

(3) FALSE. VTAB(16.8) has the same effect as the statement VTAB(16).

(4) TRUE. The computer will vertically TAB the five rows required by the VTAB statement, then will vertically TAB an additional two rows (which is equal to 80 columns) before horizontally TABbing 20 additional columns. So this statement is the equivalent of VTAB (7): PRINT TAB (20) "HELLO".

(5) FALSE. The NORMAL mode is white letters on black screen.

ADDITIONAL PRACTICE PROGRAMS

1. Write a program to accept information from a user and print a standard check such as the one that follows:

```
                                Date 5/5/77
   Pay To The Order of: John Doe
   Amount: Three Hundred and no cents...$300.00

                        Signature:
```

Hint: What information is always the same? What information changes? Use the tabbing features to format the check.

2. Write a program that produces the following output.

```
                        ABC COMPANY
                  EMPLOYEE SALARY REPORT


NAME:                YEARS EMPLOYED:
PRESENT POSITION:
     STARTING SALARY:
     PRESENT SALARY:
     DATE OF LAST RAISE:
     MONTH OF NEXT STATUS REVIEW:
```

Special Instructions:

1. All data should be INPUT.
2. Top line of headings should begin on the fifth row.
3. Top two lines of headings should be centered.
4. Information should be neatly lined up using TAB statements.

ADDITIONAL PRACTICE PROGRAMS

SUGGESTED ANSWERS

1.
```
  5  REM FIRST, GET INPUTS
 10  PRINT "CHECK IS PAYABLE TO"
 20  INPUT P$
 30  PRINT "ENTER AMOUNT IN WORDS"
 40  INPUT W$
 50  PRINT "ENTER AMOUNT AS A NUMBER"
 60  INPUT N$
 70  PRINT "ENTER DATE"
 80  PRINT D$
 90  REM PRINT CHECK
100  PRINT TAB(25) "DATE: ";D$
120  PRINT "PAY TO THE ORDER OF :";P$
130  PRINT "AMOUNT: ";W$;"...";N
140  PRINT: REM AN EMPTY PRINT STATEMENT SKIPS A LINE
150  PRINT TAB(20)"SIGNATURE:"
160  END
```

2.
```
  5  REM FIRST GET ALL INFORMATION
 10  INPUT "NAME?";NAME$
 20  INPUT "YEARS EMPLOYED?"; YEARS
 30  INPUT "PRESENT POSITION?"; P$
 40  INPUT "STARTING SALARY?"; SS
 50  INPUT "PRESENT SALARY?"; PS
 60  INPUT "DATE OF LAST RAISE?"; D$
 70  INPUT "MONTH OF NEXT STATUS REVIEW?"; M$
 80  REM FORMAT HEADING:
 90  HOME
 95  VTAB (5): HTAB (14)
100  PRINT "ABC COMPANY"
110  PRINT TAB(9) "EMPLOYEE SALARY REPORT"
120  REM FORMAT AND PRINT BODY OF REPORT
130  PRINT: REM AN EMPTY PRINT STATEMENT SKIPS A LINE
140  PRINT "NAME"; NAME$; TAB(21) "YEARS EMPLOYED:";YEARS
150  PRINT "PRESENT POSITION:"; P$
160  PRINT TAB (5) "STARTING SALARY: $"; SS
170  PRINT TAB(5) "PRESENT SALARY:$"; PS
180  PRINT
190  PRINT "DATE OF LAST RAISE:"; D$
200  PRINT TAB(5) "MONTH OF NEXT STATUS REVIEW:"; M$
210  END
```

LESSON 7
LOOPING AND BRANCHING

WHAT HAVE YOU LEARNED?

Now that you've completed Lesson 7 on the computer, you have a much better idea of what programming is all about!

You've seen examples of simple FOR-NEXT loops and learned how to create them to fulfill your own programming needs. You also saw examples of the GOTO statement, which allows you to jump over sections of your program without returning to them, and the GOSUB statement, which allows you to jump to another part of your program and then return to the line immediately following the jump.

As Alice said in Wonderland, curiouser and curiouser!

LET'S TAKE A CLOSER LOOK:

(1) The FOR-NEXT Loop:

There are many occasions in programming when it is necessary to use the same set of instructions over and over. One way of dealing with this type of problem is to program for each set of circumstances requiring these instructions. However, a much more convenient and versatile approach to the handling of repetitive tasks is to use the FOR-NEXT loop.

The FOR-NEXT loop allows you to perform a part or all of your program many times without actually repeating the lines of code themselves.

Take a look at the following program:

```
100  PRINT "ENTER NUMBER OF HOURS"
110  INPUT HOURS
120  PRINT "ENTER PAY RATE"
130  INPUT R
140  PRINT "TOTAL PAYCHECK: "; R * HOURS
150  END
```

This program asks the user to enter the number of hours worked and the rate of pay and then calculates the paycheck using this information.

If you were to use this program to process 10 paychecks, you would have to run the program 10 times. To process 100 paychecks you would have to run the program 100 times. The FOR-NEXT loop, however, makes it possible to do this type of processing much more conveniently.

Let's take a look at the same program with two lines added:

```
 90  FOR I = 1 TO 10
100  PRINT "ENTER NUMBER OF HOURS"
110  INPUT HOURS
120  PRINT "ENTER PAY RATE"
130  INPUT R
140  PRINT "TOTAL PAYCHECK:"; R * HOURS
150  NEXT I
160  END
```

Notice the two new lines, 90 and 150. These lines work together to cause the computer to process whatever lines are between them 10 times.

Line 90 declares a variable I and gives it a starting value of 1. Line 90 also tells the computer to stop processing the lines in between after I reaches 10.

Line 150 tells the computer to increment I by 1 and then go back to the beginning of the loop (line 100).

As you learned, the format of the FOR statement is:

```
FOR VARIABLE = START TO FINISH
```

Let's break this statement down:

```
VARIABLE is any legal variable
START is any integer
FINISH is any integer
```

Would this loop do exactly the same thing if line 90 read:

```
90  FOR I = 11 to 20
```

If you said "yes", you're right, since 11 to 20 will give us the same ten times through the loop as 1 to 10 or 1001 to 1010.

Here is another example of the FOR-NEXT loop:

```
10  FOR R = 2 to 3
15  PRINT "R STARTS AT 2 AND FINISHES AT 4"
20  NEXT R
25  END
```

This loop will cause line 15 to be executed twice. If this program were run, the output would look as follows:

```
R  STARTS AT 2 AND FINISHES AT 4
R  STARTS AT 2 AND FINISHES AT 4
```

R's initial value is 2 when line 15 is executed. After executing line 15 for the first time, the computer reads line 20 and increments the value of R by 1. The computer then jumps back to the first line within the loop- line 15.

At this point, since the value of R is 3, line 15 is executed again. Now, however, line 20 causes R to become 4. Since this is greater than the end value of the loop, the computer leaves the loop and continues to the next sequential statement which, in this case, is the END statement.

The FOR-NEXT loop has one additional feature which we have not yet discussed. Look at this statement:

```
For variable = start to finish STEP INTEGER
```

To this point, we have been assuming that whenever the NEXT statement is executed, the value of the loop variable is incremented by 1.

This is true unless we specify otherwise. How can we do so? By telling the computer the amount that we want to increment the variable each time through the loop.

The following program demonstrates this feature:

```
10  FOR J = 10 to 20 step 2
20  PRINT J;" ";
30  NEXT J
40  END
```

The output of this program will look like this:

```
10 12 14 16 18 20
```

On each pass through the loop, we have told the computer to add 2 to variable J. Thus, on the first pass, the initial value 10 is printed. Then J is incremented by 2 to 12, and on the second pass through, 12 is printed. This process continues until we reach or exceed the end value. After 20 is printed, J is incremented to 22, and the computer passes to the next statement after the loop and ENDs the program.

Because the FOR-NEXT loop is so important to your understanding of BASIC programming, we strongly recommend that you do all of the computer exercises at the end of this chapter.

(2) The GOTO statement:

The GOTO statement is Applesoft BASIC's way of allowing you to branch or jump to a specific line number.

```
10  GOTO 30
20  PRINT "APPLE"
30  PRINT "COMPUTER"
```

In this example, the computer is instructed to go directly from line 10 to line 30 without reading the line between, line 20. When this program is run, the output looks as follows:

```
COMPUTER
```

The GOTO statement does nothing more than tell the computer to skip to a specified line in your program and continue execution at that point.

```
10  PRINT "WELCOME TO"
20  GO TO 40
30  PRINT "LESSON 7"
40  PRINT "THE GOTO STATEMENT"
```

In the above program, the computer is instructed to jump to line 40, skipping line 30. The output of this program is therefore:

```
WELCOME TO
THE GOTO STATEMENT
```

Unless you are quite confident- and competent- in writing programs, it is recommended that you always jump forward in your program, since backward jumps may present unexpected problems. One of the computer exercises at the end of the chapter, exercise J, illustrates one such problem.

(3) The GOSUB/RETURN statements:

As you saw in Lesson 7, the GOSUB statement is similar to the GOTO statement, but with one important difference.

As with GOTO, GOSUB instructs the computer to jump to a specific line in the program, but the GOSUB remembers the line that it jumped from. When a RETURN statement is encountered, the computer jumps back to the line after the GOSUB statement.

In effect, the subroutine that is created by the GOSUB and RETURN is a program within your program. Its major advantage in programming is that it can be used within a large program to call a sequence of steps that is needed more than once- and each time the sequence is called, the RETURN statement sends the computer back to the line following the call of the GOSUB after the lines within the subroutine are executed.

Let's look at an example:

```
 5 FOR I = 1 TO 3
10  INPUT "CHOOSE A NUMBER "; A
15  INPUT "SELECT ANOTHER NUMBER "; B
20  GOSUB 35
25  NEXT I
30  END
35  PRINT A;" DIVIDED BY ";B
40  PRINT "IS ";A/B
45  RETURN
```

Line 5 is the start of a loop that will repeat three times.

Lines 10 and 15 accept two numbers as INPUT.

Line 20- GOSUB 50- sends the computer to the beginning of a subroutine on line 50.

Lines 35 and 40 are the body of the subroutine. They print the numbers chosen by the user and the result of the division of the first by the second.

Line 45, the RETURN statement, causes the computer to return to the line immediately following the GOSUB statement, line 40.

Line 40, the NEXT statement, will cause the computer to repeat the loop.

The loop will repeat three times and then fall through to the END statement at line 30. This is your first experience with the use of an END statement in the body of a program. Study it carefully so that its use is clear to you.

Subroutines should always be placed at the end of a program. The reason for this is that if the computer falls into a subroutine that is not called by a GOSUB, it encounters a RETURN statement that it cannot execute, since it has no line to return to!

Let's rewrite the previous program to make it incorrect in order to illustrate this point:

```
10  INPUT "CHOOSE A NUMBER "; A
20  INPUT "SELECT ANOTHER NUMBER "; B
30  PRINT A; " DIVIDED BY "; B
40  PRINT "IS "; A/B
50  RETURN
60  GOSUB 30
70  GOTO 10
```

In this example, lines 10 and 20 are the same as in the correct example.

Line 30 is really the beginning of our subroutine, but it is an executable line to the computer, and so lines 30 and 40 are executed.

When the computer reaches line 50, it attempts to RETURN, but since it has not yet encountered a GOSUB, it has no line to return to. Therefore, it prints an error message:

```
RETURN WITHOUT GOSUB IN 50
```

and halts execution of the program. Lines 60 and 70 are never reached.

Let's summarize the important things to know about the GOSUB/RETURN statements:

1. Use the GOSUB to access a line or series of lines that are used more than once in a program.

2. Every subroutine must end with a RETURN statement.

3. Place your subroutines at the end of your program so that the computer does not accidentally "fall" into them.

Subroutines are a powerful programming tool. They not only save memory space by making it possible for you to avoid repetition within your program, but they also often make it possible to break complex programs into more readable and understandable pieces. Be sure to work on each of the exercises at the end of this chapter until you thoroughly understand FOR-NEXT loops, GOTO statements and GOSUB/RETURN statements.

COMPUTER EXERCISES

PREDICT THE EXPECTED OUTPUT FOR EACH PROGRAM; THEN ENTER AND RUN THE PROGRAM:

(A)
```
NEW
10 FOR I = 1 TO 3
20 PRINT I
30 NEXT I
40 PRINT I
```

(B)
```
NEW
10 A = 4
20 FOR B = 3 TO 5
30 A = A + 2
40 NEXT B
50 PRINT A
```

(C)
```
NEW
10 FOR VAR = 1 TO 10 STEP 3
20 PRINT VAR
30 NEXT VAR
```

(D)
```
NEW
10 FOR A = 1 TO 5
20 PRINT "ENTER
   QUANTITY"
30 INPUT Q
40 PRINT "ENTER PRICE"
50 INPUT P
60 PRINT"REVENUE = ";
   Q * P
70 NEXT A
```

(E)
```
NEW
10 FOR P = 10 TO 1 STEP-1
20 PRINT P
30 NEXT P
```

(F)
```
NEW
10 A = 3
20 FOR I = 1 TO A
30 PRINT I;" SQUARED
   IS ";I * I
```

(G)
```
NEW
10 GOTO 30
20 PRINT "NEVER EXECUTED"
30 PRINT "I WAS HERE"
40 END
```

(H)
```
NEW
10 R = 5
20 GOTO 40
30 R = 10
40 PRINT R
```

(I)
```
NEW
10 N = 10: I = .10
20 GOSUB 60
30 N = 8: I = .06
40 GOSUB 60
50 END
60 TI = N * I
70 PRINT "BASE", "RATE", "DUE"
80 PRINT N, I, TI
90 PRINT
100 RETURN
```

You can stop the following program by holding down the CNTL key and pressing the letter C:

(J)
```
10 PRINT "INFINITE LOOP"
20 GOTO 10
```

CHAPTER 7

EXERCISE ANALYSIS

(A) OUTPUT: 1
2
3
4

I is set to 1 to begin the loop and is incremented by 1 each time through the loop. When I reaches 4, the loop ends, and the PRINT statement in line 40 is executed.

(B) OUTPUT: 10

A is 4 upon entering the loop. On the first pass through the loop B is 3, and A becomes 6. On the second pass B is 4, and A becomes 8. On the third pass B is 5, and A becomes 10. When B becomes 6, the loop ends, and the program prints the final value in A.

(C) OUTPUT: 1
4
7
10

VAR is set to 1 to begin the loop and is incremented by 3 on each pass through until the end value (10) is reached or exceeded. The PRINT statement inside the loop causes the value of VAR to be printed on each pass through.

(D) OUTPUT: varies with INPUT

This program requires the entry of 5 sales quantities and prices and then calculates the revenue for each set of INPUT.

(E) OUTPUT: 10
9
8
7
6
5
4
3
2
1

This program presents a FOR-NEXT loop with a negative step cycle. The computer carries out subtraction as easily as it carries out addition.

(F) OUTPUT: 1 squared is 1
2 squared is 4
3 squared is 9

This program uses a variable to set the end point of the loop, then prints the value of I, a literal, and then multiplies I by itself and prints it.

(G) OUTPUT: I WAS HERE

This program begins by moving directly to line 30 and executing each line from that point.

(H) OUTPUT: 5

This program sets the value of R to 5 and then goes directly to line 40 and prints the value of R.

(I) OUTPUT:

BASE	RATE	DUE
10	.10	1.00

BASE	INTEREST RATE	INTEREST PAID
8	.06	.48

(This program accepts two values, N and I and uses a subroutine to multiply them and format the way in which they will be printed. It then changes the values of N and I and returns to the subroutine to duplicate the process.)

(J) This program is an infinite loop. As long as the program is being executed it will continuously return to line 10 to print INFINITE LOOP. One of the major reasons to avoid GOTO statements that jump back in a program is that using such statements increases the possibility that your program will enter an infinite loop.

ADDITIONAL PRACTICE PROGRAMS

1. Write a program that prints a student's report card average. The program should ask for the student's name and 5 grades. The program should print out the student's name and average.

2. Suppose Mr. Jones starts with 1 penny. How much money will he have at the end of 36 days if he doubles his money each day?

3. Design a program that calculates population growth. Accept a base population, a rate of growth, and a number of years and print out the population for each year of the sequence.

ADDITIONAL PRACTICE PROGRAMS

SUGGESTED ANSWERS

1.
```
10 PRINT "ENTER NAME:";
20 INPUT NA$
30 FOR TESTS = 1 TO 5
40 PRINT "ENTER GRADE:";
50 INPUT GRADE
60 TTL = TTL + GRADE
70 NEXT TESTS
80 PRINT TAB(15) NA$
90 PRINT:PRINT
100 PRINT "AVERAGE: ";TTL/5
110 END
```

2.
```
10 REM THIS PROGRAM CALCULATES TOTAL MONEY AFTER 36 DAYS
20 M = .001: REM START WITH ONE PENNY
30 FOR DAYS = 1 TO 36
40 M = M * 2
50 NEXT DAYS
60 PRINT "TOTAL AMOUNT AFTER 36 DAYS = ";M
70 END
```

3.
```
10 REM THIS PROGRAM CALCULATES POPULATION GROWTH
20 INPUT "BASE POPULATION"; BASE
30 INPUT "RATE OF GROWTH"; R:REM DON'T USE RATE AS A
   VARIABLE-IT HAS THE RESERVED WORD "AT" EMBEDDED IN
   IT.
40 INPUT "NUMBER OF YEARS "; YEARS
50 PRINT:REM SKIP A LINE BEFORE BEGINNING TABLE
60 REM PRINT COLUMN HEADINGS AND BASELINE DATA
70 PRINT "YEAR", "NEW", "YEARLY"
80 PRINT    , "POP", "INCREASE"
90 PRINT
100 REM LOOP TO CALCULATE AND PRINT LIST
110 P = BASE: REM USE VARIABLE P FOR NEW POPULATION
120 FOR LOOP = 1 TO YEARS
130 I = P * R: REM I IS YEARLY INCREASE
140 P = P + I
150 PRINT LOOP, P, I
160 NEXT LOOP
170 END
```

*NOTE: If you return to this program after completing lesson 10, you will be able to eliminate as many places after the decimal point as you want to when the list is printed.

LESSON 8
CONDITIONAL TESTING

WHAT HAVE YOU LEARNED?

Up to this point, you have used your Apple to perform very simple tasks in programming, but in Lesson 8, you discovered the way in which you can program the computer to make decisions.

Using the IF-THEN statement to branch to subroutines or to go to alternate branches of your program involvestechniques that will come with practice. The longest journey begins with the first step!

LET'S TAKE A CLOSER LOOK:

(1) IF-THEN:

The IF-THEN statement is one of the most important ideas in BASIC programming. The statement takes the form:

IF <expression> THEN <action>

The <expression> is a comparison that the computer makes between variables in its memory. The following are examples of valid expressions:

```
IF A = B  THEN
IF A$ <> "CONTINUE" THEN
IF A+5 < B THEN
IF RENT > BANKACCOUNT THEN
IF GUESS = PREDICT THEN
```

As long as the variables are of the same type (remember: you can't compare a string to a numeric!), you can use them for the comparison that forms the basis of the IF-THEN decision.

The other half of the IF-THEN statement, the THEN, can be any valid BASIC command.

Let's look at some examples.

If A = B THEN GOTO 180 (If A is equal to B THEN the program transfers to line 180, otherwise, it continues to the next line.)

IF A$ <> "CONTINUE" then END (IF A$ holds anything but the word "continue", THEN execution of the program will end; otherwise the computer moves to the next line).

IF A + 5 < B THEN B = B-5 (IF A plus 5 is less in value than B, THEN subtract 5 from B; otherwise, continue to the next line).

IF RENT > BANKACCOUNT THEN PRINT "HOLD THAT CHECK." (IF the amount in variable RENT is greater than the amount in variable BANKACCOUNT THEN the statement is printed. Otherwise, the next line is executed).

IF GUESS = PREDICT THEN GOSUB 12000. (IF the number in variable GUESS is equal to the number in variable PREDICT THEN go to a subroutine beginning on line 12000; otherwise, continue to the next line).

You can greatly increase the power and value of your BASIC program by using IF-THEN statements. The ability of the computer to make decisions based on comparisons is one of the major reasons for the speed with which the computer age has burst upon us. Let's take a look at some examples:

```
 5  OH = 0: HOURS = 0:REM OH = OVERTIME HOURS
10  OTR = 4.87:REM OVERTIME RATE OF PAY
20  R = 3.25:REM REGULAR RATE OF PAY
30  INPUT "ENTER HOURS WORKED"; HOURS
40  IF HOURS > 40 THEN OH = HOURS - 40
50  IF HOURS > 40 THEN HOURS = 40
60  PRINT "NORMAL PAY: "; HOURS * R
70  PRINT "OVERTIME PAY: "; OH * OTR
80  PRINT "TOTAL PAY:";HOURS * R + OH * OTR
90  GOTO 5
```

This program can be used to calculate the amount of an employee's paycheck. Let's analyze it line by line:

Line 5 wipes out any information that may have been in variables OH and HOURS from previous employees.

Line 10 sets the variable OTR equal to 4.87. This is the overtime rate of pay.

Line 20 sets the variable R equal to 3.25. This is the rate of regular pay.

Line 30 asks the user to INPUT the number of hours worked and stores the information in a variable called HOURS.

Line 40 tells the computer to compare the variable HOURS to the number 40. IF HOURS is greater than 40 THEN overtime hours (variable OH) are calculated by subtracting 40 from HOURS. IF HOURS is not greater than 40, this line is not executed.

Line 50 will be executed only if line 40 is executed since it looks for the same condition as does line 40. This line tells the computer to set hours equal to 40 if, in fact, the number of hours is greater than 40. Since line 40 has already calculated overtime hours, this line makes it possible to calculate normal pay. If line 50 were not present, normal pay- which is calculated by multiplying the hours by the rate- would include overtime hours as well as normal hours.

Line 60 calculates the normal pay, which is a product of R times HOURS (up to 40 hours).

Line 70 calculates overtime pay, which will be zero for cases in which HOURS is not greater than 40.

Line 80 calculates and prints out the total pay for each employee.

Line 90 sends the program back to line 5 to begin calculations for the next employee.

If you are encountering difficulty in understanding the function of any line in this program, we suggest that you review the lesson that explains that line. The guide below will help you to find these lessons easily.

LINES	TOPIC	LESSON
10 and 20	Variables	3
30	INPUT	5
40 and 50	IF/THEN	8
60, 70 and 80	PRINT statements	2
90	GOTO	7

Let's look at another program that uses the IF-THEN statement:

```
10  PRINT "WHAT IS 12 - 6"
15  INPUT AN
20  IF AN = 6 THEN PRINT "VERY GOOD"
25  IF AN <> 6 THEN PRINT "THAT'S INCORRECT"
30  END
```

This short program checks to see if the user got the right answer to the question: "WHAT IS 12 - 6?" If INPUT AN is 6, then the comparison in line 20 (AN = 6) is true and the PRINT statement is executed. Obviously if this is the case, the comparison in line 25 (AN <> 6) is false, and so the PRINT statement in line 25 is not executed.

On the other hand, if the answer given is not 6, then the comparison in line 20 (AN = 6) is false and that line is not executed while the true comparison in line 25 (AN<>6) causes the computer to execute the PRINT statement "THAT'S INCORRECT."

(2) RELATIONAL OPERATORS:

As you learned in Lesson 8, there are six relational operators. The following table lists them and their meanings:

1. = EQUAL TO
2. > GREATER THAN
3. < LESS THAN
4. <= LESS THAN OR EQUAL TO
5. >= GREATER THAN OR EQUAL TO
6. <> NOT EQUAL TO

These are all of the possible comparisons that can be used in the <expression> part of the statement:

IF <expression> THEN <action>.

To reiterate, the <action> part of this statement can be any legal BASIC statement. Let's look at another example:

```
20  INPUT "WHAT IS 100/20?"; AN
30  IF AN = 5 THEN GO TO 60
40  PRINT "THAT'S INCORRECT"
50  GO TO 20
60  PRINT "VERY GOOD"
70  END
```

In this example, if the answer is correct, the computer jumps to line 60, prints a message and ends. If it is not correct, the program does not execute line 30, but rather continues to line 40, where a different message is printed. The computer then is sent back to line 20 where the sequence begins again. The user may continue to work on this program until he or she arrives at the right answer.

This is a good time to note that if the result of the IF comparison is true, the entire line is executed, so if we put more statements on a line than the IF/THEN comparison, each of the additional statements will be executed. Take, for example, the following program:

```
100  A = 110
110  IF A > 9 THEN A = A/2: B = A: PRINT B
120  END
```

In this example, since A is greater than 9, the statement is true. Therefore, A is divided by two, B is set equal to A, and B is printed.

If the comparison evaluated to "false," then none of line 110 would have been executed.

EXTRA INFORMATION

THE FORM OF THE IF-THEN STATEMENT:

When the IF-THEN statement is used to branch to a line, as in:

```
100 IF A > B THEN GOTO 150
110 IF A$ = "HELLO" THEN GOTO 200
```

Applesoft BASIC will allow us to omit the word THEN from the line. The following are legal variations of the above lines:

```
100 IF A > B GOTO 150
110 IF A$ = "HELLO" GOTO 200
```

As an alternative, we can omit the word GOTO if we include the word THEN, as follows:

```
100 IF A > B THEN 150
110 IF A$ = "HELLO" THEN 200
```

Computer Exercises

1.) At the computer, enter and execute the following program:

```
10  FOR J = 10 TO 1 STEP -1
20  PRINT J
30  IF J = 1 THEN PRINT "BLAST OFF!"
40  NEXT J
50  END
```

2.) How can we rewrite the above program without using an IF-THEN statement?

3.) What will be printed below when this program is run? (The answer is at the bottom of the page).

```
100  LET B = 15
120  LET C = 20: LET D = 50
140  IF B + 20 > C THEN GOTO 180
160  PRINT"LINE 160"
180  GOTO 220
200  GOTO 300
220  PRINT "LINE 220"
240  IF C * 2 + 10 >= 50 GOTO 260
260  PRINT "LINE 260"
280  GOTO 200
300  PRINT "LINE 300"
320  END
```

ANSWER TO QUESTION 2

```
10 FOR J = 10 T1 STEP -1
     20 PRINT J
     40 NEXT J
     45 PRINT"BLAST OFF!"
     50 END
```

ANSWER TO QUESTION 3

```
     LINE 220
     LINE 260
     LINE 300
```

MORE PROBLEMS TO COME.

ADDITIONAL PRACTICE PROGRAMS

1. A. Write a program to simulate a candy machine. If the user enters more than 30 cents, print the change he should get. If the user enters less than 30 cents, tell him/her how much more to enter. If the user enters the correct amount, display 'THANK YOU.'

```
sample:  ENTER 30 CENTS PLEASE
         ?20
         ADD 10 CENTS PLEASE
```

B. Modify the above program to keep on executing until the right amount is entered.

```
sample output:   ENTER 30 CENTS PLEASE
                 ?15
                 ADD 15 CENTS PLEASE
                 ?5
                 ADD 10 CENTS PLEASE
                 ?10
                 THANK YOU
```

2. Write a program which accepts numbers until a 7 is entered. When a 7 is entered, print the message 'FINALLY.'

3. Write a program which does the following:

   A. Accepts numbers (positive or negative) until a 0 is entered.

   B. Keeps a running total of the numbers.

   C. If the total is over 100 at any point, sets the total to 0 and prints 'OVER.'

   D. If the total is less than 0 at any point, sets the total to 0 and prints 'UNDER.'

```
sample:  ENTER NUMBER ?80
         ENTER NUMBER ?-90
         UNDER
         ENTER NUMBER ?101
         OVER
```

4. The ABC Company is a wholesaler of women's dresses. Its policy is to charge the following:

$39.88/dress for over 50 dresses
$49.88/dress for between 25-50 dresses
$59.88/dress for less than 25 dresses

Write a program that enables a clerk to enter the number of dresses, and have the computer calculate the total cost of the dresses.

ADDITIONAL PRACTICE PROGRAMS

SUGGESTED ANSWERS

1.

A.
```
10 PRINT "ENTER 30 CENTS PLEASE"
20 INPUT AMOUNT
30 IF AMOUNT < 30 THEN PRINT "ADD";30-AMOUNT
40 IF AMOUNT > 30 THEN PRINT "YOUR CHANGE IS";
   AMOUNT-30
50 IF AMOUNT = 30 THEN PRINT "THANK YOU"
```

B.
```
 5 NA = 30
10 PRINT "ENTER 30 CENTS PLEASE"
20 INPUT AMOUNT
30 TTL = NA - AMOUNT
40 IF TTL > 0 THEN PRINT "ENTER ";TTL;" CENTS PLEASE"
45 NA = TTL:GOTO 20
50 IF TTL = 0 THEN PRINT "THANK YOU"
60 IF TOTAL < 0 THEN PRINT "YOUR CHANGE IS ";-1 * TTL
```

2.
```
10 PRINT "PLEASE ENTER A NUMBER."
20 INPUT NUMBER
30 IF NUMBER <> 7 THEN GOTO 10
40 PRINT "FINALLY"
50 END
```

3.
```
 5 TTL = 0: REM INITIALIZE COUNTER
10 INPUT "ENTER NUMBER?;NUMBER
20 IF NUMBER = 0 THEN GOTO 100
30 TTL = TTL + NUMBER
40 IF TTL > 100 THEN GOSUB 200
50 IF TTL < 0 THEN GOSUB 300
60 GO TO 10
200 PRINT "OVER"
210 TTL = 0
220 RETURN
300 PRINT "UNDER"
310 TTL = 0
320 RETURN
```

4.
```
10 PRINT "ENTER NUMBER OF DRESSES PURCHASED"
20 INPUT N
30 IF N > 50 THEN PRICE = 39.88
40 IF N <= 50 THEN PRICE = 49.88
50 IF N < 25 THEN PRICE = 59.88
60 PRINT "TOTAL COST OF ALL DRESSES IS $"; N * PRICE
70 END
```

LESSON 9
STRING HANDLING TECHNIQUES

WHAT HAVE YOU LEARNED?

In Lesson 9 you were introduced to some of the more subtle aspects of string handling that Applesoft BASIC makes available to you.

You learned how to use the LEN function to count the number of characters in a string and the complementary functions VAL and STR$ which respectively convert a string to a number and a number to a string.

You also learned how to separate strings into smaller parts using the LEFT$, RIGHT$ and MID$ functions.

Although the value of these functions may not be immediately apparent to you, they are useful programming tools that will increase in value as your understanding of and experience with computer programming grows.

LET'S TAKE A CLOSER LOOK:

(1) The LEN function:

Quite simply, the LEN function (short for length) counts the number of characters in a string. Let's look at a few examples:

```
LEN ("WATER") IS 5
LEN ("HEALTHY") IS 7
LEN ("IT") IS 2
```

The LEN function can be used with string variables as well as with literals, as the following program demonstrates:

```
10  Q$ = "WINDOW"
20  PRINT LEN (Q$)
```

The output, of course, would be 6.

How can we use the LEN function in a meaningful program? Suppose we want to keep a list of telephone numbers. We can have the numbers INPUT into a string variable and saved in the computer's memory. Obviously, all telephone numbers have 7 digits. Therefore, if we accidentally type in a number that does not have 7 digits, we would want the computer to inform us that we have not entered a valid telephone number:

```
10  PRINT "ENTER TELEPHONE NUMBER"
20  INPUT TEL$
30  IF LEN (TEL$) <> 7 THEN PRINT "INVALID NUMBER.
    PLEASE CHECK AND REENTER."
40  GO TO 10
```

In line 30 we tell the computer to count the number of characters in the variable TEL$ and to warn us if the number is not 7.

To summarize, the LEN function is a useful tool for checking string data when the length of the string is uniform, such as in Social Security numbers, and telephone numbers.

Do other uses for the LEN function suggest themselves to you?

(2) The Value of VAL and STR$

VAL is used to convert a string to a numeric value. In discussing LEN, we introduced the idea of entering numeric data in strings, so that it might be checked before attempting to use it. Let's look at a simple example:

```
10  INPUT "TYPE A NUMBER BETWEEN 1 AND 100"
20  Y = VAL (Y$)
30  IF Y = 0 THEN PRINT "INVALID INPUT"
```

Here, we have taken a numeric value as input and placed it in a string variable. (There is no harm in doing this as long as we can later retrieve its numeric value.) The VAL function in line 20 instructs the computer to convert the string to its actual numeric value. If the string is not numeric (for instance, if the user had accidentally entered the characters V5 or ZC) the VAL function returns a value of 0. Line 30 alerts the user to the fact that the input in line 10 is not numeric if that is the case.

The STR$ function is the complement of the VAL function in that it converts numbers to strings. Thus, the following sequence:

```
10  Y = 123
20  Y$ = STR$(Y)
```

converts the number 123 to the string 123.

Why is this function useful? On occasion we may want to bring two numbers together. For instance, if we were doing an inventory in which the first two digits of an item number referred to its room location, we might want to concatenate the area number to the item number on an inventory list.

Thus, we would want item number 145 in area 47 to be numbered 47145. We cannot do this using numerics, because adding them will not result in this numeric arrangement. However, if we have them as numbers, we can change them to strings using STR$, and as you learned in Lesson 3, we can then concatenate them easily.

```
100  LET ITEM = 145:LET AREA = 47
110  ITEM$ = STR$(ITEM):AREA$ = STR$(AREA)
120  INV$ = AREA$ + ITEM$
```

(3) The String Functions:

In Lesson 9 we presented 3 string functions: LEFT$, RIGHT$ and MID$. These functions are used to extract parts of a string from a complete string.

LEFT$ is used to extract a certain number of characters starting at the leftmost character of the string. Take a look at the following example:

```
PRINT LEFT$ ("SEWING", 3)

SEW
```

The above statement tells the computer to extract 3 characters starting from the left of the string "sewing."

RIGHT$ works in the same way as LEFT$, except that it starts at the rightmost character of a string. Using the same example as above, you can see how this works:

```
PRINT RIGHT$ ("SEWING", 3)

ING
```

Notice that RIGHT$ extracted the last three characters whereas LEFT$ extracted the first three characters.

MID$ also extracts characters from a string; however you must specify the starting point as well as the number of characters. Let's look at an example:

```
PRINT MID$ ("STRING", 3,4)
```

RING

The above statement directs the computer to look at the literal "STRING", to begin extracting characters at the third character from the left and to extract four characters. Let's take a closer look at this statement:

```
                        --- Tells computer to begin at third
                        !   character from left
PRINT MID$ ("STRING", 3, 4)
                           !
                           --- Tells computer to extract four
                               characters
```

Are you ready to check your understanding of string function statements? See if you can predict what each of these will produce in output. The answers are on page 87.

```
PRINT LEFT$ ("PHOTO", 3)
PRINT LEFT$ ("HELLO", 2)
PRINT RIGHT$ ("PHOTO", 2)
PRINT RIGHT$ ("AUTHOR", 4)
PRINT MID$ ("WINDY", 2,3)
PRINT MID$ ("PHOTOGRAPHY", 6,5)
```

There are many useful applications for the string functions. To demonstrate one common use, let's write a department store inventory program.

Many stores use a code to describe each item they carry. Let's take the XYZ Co., a sporting goods retailer. The inventory is received from 3 sources, designated 1, 2,and 3. The store code for each item is as follows:

```
                ---Stock Location
           13492.62
            ----Item Number
           -Source of Inventory
```

If the above were an actual inventory number for XYZ company, it might tell us that item 3492 is a tennis racquet supplied by Source #1 and stocked in area 62.

Now let's write a small program that will accept the inventory number for any product and print a summary as follows:

XYZ Co.

INVENTORY SUMMARY

ITEM NO.	LOCATION	SOURCE
3492	62	1
8741	35	3
7265	12	2

To develop any program we need to answer three questions:

1. What must be input to the program?
2. What processing is necessary?
3. What output do we want?

In this case, the data to be input consists only of the store code for each item, since it has embedded within it the item number, location, and source code.

The processing that is required is the breaking up of this store code into its 3 parts, so that they can be printed.

The output is the above summary.

Let's write the program and discuss it line by line:

```
            CODE                          COMMENTS

100 PRINT TAB(16); "XYZ CO."          (Prints first line of heading)
110 PRINT TAB(11);"INVENTORY          (Prints second line of
    SUMMARY"                           heading)
120 PRINT "ITEM NO.",                 (Prints & spaces third line)
    "LOCATION","SOURCE"
130 INPUT "ITEM CODE";CODE$           (Inputs store code)
140 IF CODE$ = "0" THEN               (Ends program if 0 is input.)
    GOTO 200                           This is how we tell the
                                       computer that we are done.)
150 PRINT MID$(CODE$,2,4),            (Extracts and prints item
                                      number)
160 PRINT RIGHT$(CODE$, 2),           (Extracts and prints stock
                                       location)
170 PRINT LEFT$ (CODE$, 1)            (Extracts and prints source
                                      code)
180 GOTO 130                          (Return for next store code)
200 END
```

Think you've got it? It's not that hard after all, is it? If you want to review, though, we suggest that you first return to Lesson 9 at the computer.

COMPUTER EXERCISES

1.) What will the following display?

```
A.  PRINT LEFT$ ("DOG",1)
B.  PRINT LEFT$ ("BEDROOM",3)
C.  PRINT RIGHT$ ("CAPE",3)
D.  PRINT RIGHT$ ("CANDY BAR",3)
E.  PRINT MID$ ("STAND",3,3)
F.  PRINT MID$ ("COFFEE",2,3)
```

2. What is VAL ("HELLO")?

3. Can STR$ (123) be concatenated to a string?

4. Write a program to extract the first and last characters from a string, and concatenate them.

5. Write a program which asks for a string and prints each letter of the string on a separate line.

```
SAMPLE:
     ENTER A STRING
     ?BOAT
     B
     O
     A
     T
```

6. What do the following display?

```
A.  LEN("GOLF")
B.  LEN("")
C.  LEN(THE GOLFER")
```

CHAPTER 9

EXERCISE ANSWERS AND ANALYSIS

1. A) D B) BED C) APE D) BAR E) AND F) OFF

2. The VAL of all strings is 0.

3. YES. STR$ converts a numeric to a string. One of the primary reasons that you might want to do this in a program is to concatenate numeric input.

4. The following program is only one suggestion to put you on the right track. The program that you have written to answer this question may be entirely different, but equally correct.

```
10  INPUT A$
20  LET B$ = LEFT$(A$,1) + RIGHT$(A$,1)
30  PRINT B$
```

5. We won't write this program for you, but we'll help you get started with a few hints:

A. Use a FOR-NEXT loop in which the left$ function uses the loop variable: PRINT MID$(A$,I,1) where A$ is the string and I is the loop variable. B. Place your print statement within the loop.

6. A) 4 B) 0 C) 10 (The computer counts blanks as well as characters within a string).

ANSWERS TO QUESTIONS ON PAGE 84:

STATEMENT	OUTPUT
Print LEFT$ ("PHOTO",3)	PHO
Print LEFT$ ("HELLO",2)	HE
Print RIGHT$ ("PHOTO", 2)	TO
Print RIGHT$ ("AUTHOR",4)	THOR
Print MID$ ("PHOTOGRAPHY",6,5)	GRAPH

ADDITIONAL PRACTICE PROGRAMS

1. Accept from the user any string of characters and display those characters in reverse.

```
sample:  INPUT A STRING PLEASE
         ?HERE IS A STRING
         GNIRTS A SI EREH
```

HINT: a good way to solve this problem is by using the LEN function and a FOR-NEXT loop with a negative STEP.

2. a. Write a program which accepts a string and determines if it contains the letter 'A'.

   b. Modify the above program to count the number of "A"'s in the string.

ADDITIONAL PRACTICE PROGRAMS

SUGGESTED ANSWERS

1.
```
  10 PRINT "PLEASE ENTER A STRING"
  20 INPUT A$
  30 REM START AT LAST CHARACTER AND WORK BACKWARDS
  40 FOR J = LEN(A$) TO 1 STEP -1
  50 PRINT MID$(A$,J,1);:REM SEMI-COLON KEEPS ALL
CHARACTERS ON THE SAME LINE
  60 NEXT J
```

2.

A.
```
  10  PRINT "PLEASE ENTER A STRING"
  20  INPUT S$
  30  FOR I = 1 TO LEN(S$)
  40  IF MID$(A$,I,1)="A" THEN GOTO 70
  50  NEXT I
  60  GOTO 80
  70  PRINT S$;" CONTAINS AN A"
  80  END
```

B.
```
   5  C = 0
  10  PRINT "PLEASE ENTER A STRING"
  20  INPUT S$
  30  FOR I = 1 TO LEN(S$)
  40  IF MID$(S$,I,1) = "A" THEN C = C + 1
  50  NEXT I
  60  PRINT S$:" CONTAINS ";C;" A'S"
  70  END
```

LESSON 10
INTRODUCTION TO MATHEMATICAL FUNCTIONS

WHAT HAVE YOU LEARNED?

In Lesson 10, you learned how to use two mathematical functions: INT(X) and RND(1).

You saw that the INT(X) function is used to remove the decimal portion of a real number, thereby leaving you with an integer.

You also saw that the RND(1) function is a random number generator.

There are many uses for both the INT(X) and RND(1) functions. In this chapter, we'll explore a few.

LET'S TAKE A CLOSER LOOK:

(1) The Making of an INTeger:

As we are using the word, a function is nothing more than a BASIC statement that returns a value. The INT function is used to return an integer value. Its form is:

```
Let <numeric variable> = INT (<numeric variable or
                                number>)
```

In computerese, the variable or number in parentheses (that is, the variable that is being worked on by the computer) is called an argument. Every integer statement must have a valid argument.

The statement:

```
LET X = INT(X)
```

is a valid statement in BASIC. On the other hand:

```
LET X$ = INT(X)
```

or:

```
LET X = INT(X$)
```

are both illegal statements. Since integers exist only in the numeric world, the INT function can be used only with numeric variables or actual numbers.

Let's look at a few other valid statements:

```
PRINT INT(W)                        <W is the argument>
LET TEMP = INT (55.5)               <55.5 is the argument>
LET AVERAGE = INT (TESTGRADES)<TESTGRADES is the argument>
```

Now that we've reviewed the form of the INT statement, let's consider the way in which you used it in Lesson 10. The INT function is used to strip a positive number of anything after the decimal point. This process is called truncation. When the INT function is used with a positive number, the computer chops off any number to the right of the decimal point in the argument. If the following statement were executed:

```
10  PRINT INT (18.6)
```

the computer would display an 18.

When INT is used with a negative number the integer less than or equal to the number will be returned. If the following statement were executed:

```
10  PRINT INT (-7.2)
```

the computer would display a -8.

Below is a table showing the value of some other numbers used with the INT statement.

Statement	Result
PRINT INT (12.2)	12
PRINT INT (-4.7)	-5
PRINT INT (-4.36)	-5
PRINT INT (33.9)	33
PRINT INT (12)	12
PRINT INT (0)	0
PRINT INT (-14.8)	-15

The INT statement can be a useful tool in processing numbers. The program below is widely used in BASIC to round to the nearest hundredth (or, if you're working with money, the nearest cent):

```
10  PRINT "INPUT A NUMBER"
20  INPUT N
30  X = (INT(N * 100 + .5))/100
40  PRINT "YOUR NUMBER IS "; X
```

Let's take a look at the statement in line 30. Assume that we have typed 32.755 in response to the INPUT statement.

Line 30 instructs the computer to carry out several arithmetical operations. It is important to recall the order in which they will be carried out:

FIRST, expressions within the parentheses are evaluated, with inner parentheses taking precedence. Thus:

```
(N * 100 + .5)
```

is evaluated first. Which takes arithmetical precedence, though, multiplication or addition?

If you said multiplication, you're right. Multiplication and division are equal to each other but always take precedence over addition and subtraction, so the inner expression evaluates as follows:

```
32.755 times 100 is 3275.5
3275.5 + .5 is 3276.0
```

SECOND, the expression within the outer parentheses is evaluated. In this case, the expression is the call of the integer function:

```
INT (3276.0)
```

The integer function truncates the expression to 3276. Finally, division by 100 takes place:

```
3276/100 is 32.76.
```

Let's look at one more example. Suppose you are writing a program to calculate test averages to the nearest tenth. Since some of your students have missed an exam or exams, you must tell the computer when you have finished entering grades for student A and want the average calculated. You also must have the computer count the number of exams in order to calculate the average properly. Think you can do it? Let's write this one together, with a line by line explanation to follow.

```
 10  NUMTESTS = 0: TTL = 0
 20  INPUT "TYPE IN A TEST SCORE.  WHEN YOU HAVE TYPED
     IN THE LAST SCORE FOR THIS STUDENT, TYPE IN A
     NEGATIVE NUMBER."; S
 30  IF S < 0 THEN GO TO 200
 40  NUMTESTS = NUMTESTS +1
 50  TTL = TTL + S
 60  GOTO 20
200  AVERAGE = TTL/NUMTESTS
210  TESTAVERAGE = (INT(AVERAGE * 10 + .5))/10
220  GOTO 10
```

Now, the line by line breakdown:

LINE 10
sets the number of tests and total score equal to zero so that we can begin calculations anew for each student.

Line 20
is our input statement.

Line 30
is our check to see if this is the end of input for this student. In line 20 we told the user to type a negative number after all data was entered for a given student, so we want to make sure that negative number isn't included as one of the test grades!

Line 40
tells the computer to increase the number of tests by 1. By this point, we have rerouted all invalid test grades, so that any grade that reaches line 50 is the real thing.

Line 50
adds the current test score to the total already accumulated.

Line 60
sends the computer back for the next input, since no other processing is to take place until all scores for this student are entered.

Line 200
computes the average.

Line 210
recalculates the average rounded off to the nearest tenth. Compare this INT statement to the one in the previous program. Note that to go to the nearest tenth we multiply and divide by 10, but for the nearest hundredth, we multiply and divide by 100. How would we round to the nearest thousandth?

Line 220
sends the computer back to start over with a new set of scores.

If you've understood all or most of this program, you've indeed come a long way in the development of your programming skills!

(2) RANDOM NUMBERS:

The RND function is, perhaps, the most unusual expression in the BASIC language, because while most instructions in programming require that the computer be predictable, RND asks the computer to be unpredictable. Unpredictable is, of course, almost a synonym for "random" and since the RND(1) function returns a random number, predictability is out of the question!

As you saw in Lesson 10, RND(1) will always return a number greater than zero, but less than 1. The following are typical random numbers generated by the Apple:

STATEMENT	RESULT
Print RND (1)	.86441729
Print RND (1)	.25311182
Print RND (1)	.99645318

Notice that the argument of RND is always (1) and that the number returned is always carried to 8 decimal places. This format is not suitable for most situations in whichrandom numbers are required, though. How, then, can we make this function useful?

We've found another use for INT!

Knowing that RND will always produce a number greater than or equal to zero and less than 1, we can derive a BASIC statement that can change that number into a form more useful to us.

For instance, if we wanted a random integer between 1 and 100:

The statement:

```
LET X = INT (RND(1)*100)+1
```

would provide us with one. Let's see how:

First, the statement within the parentheses generates a random number and multiplies it by 100.

Next, INT truncates the number to an integer.

Finally, we add 1 to eliminate the possibility that the result will be zero.

How can we generate a random number between 1 and 15?

```
LET S = INT(RND(1)*15)+1
```

How about a random number betwen -4 and 5?

```
LET J = INT(RND(1)*9)-4
```

In all of these examples, the number to the far right is the lowest number we want and the number in parenthesis that we multiply the random number by is the range.

Let's look at a program that uses random numbers:

```
 5  FOR I = 1 to 10
10  L = INT (RND(1)*100)+1
20  M = INT (RND(1)*100)+1
30  PRINT "WHAT IS ": L: "+": M
40  INPUT ANSWER
50  IF ANSWER = L+M THEN PRINT "THAT IS CORRECT!"
60  IF ANSWER < > L+M THEN PRINT "THAT'S NOT CORRECT!"
70  NEXT I
80  END
```

Line 5
tells the computer that we are going to execute this loop 10 times.

Line 10
generates a random number between 1 and 100.

Line 20
does the same.

Line 30
asks the user to add the two numbers (of course, this program would work just as well for subtraction, division, multiplication or exponentiation).

Line 40
takes the user's answer.

Lines 50 and 60
evaluate the answer and print the appropriate response.

Line 70
returns for the next problem.

Would you like to improve on this program? Try to write the code that will give the user a second chance to do the problem and will give the correct answer after the user has input two incorrect answers. Our suggested code is on page 102.

COMPUTER EXERCISES

At the computer, enter and execute the following programs:

(1)
```
10 PRINT "GUESS MY NUMBER - IT'S FROM 1 TO 10"
20 X = INT(RND(1)*10)+1
30 INPUT GUESS
40 IF GUESS < X THEN PRINT "THAT'S INCORRECT-TOO LOW"
50 IF GUESS > X THEN PRINT "THAT'S INCORRECT-TOO HIGH"
60 IF GUESS = X THEN PRINT "YOU'RE RIGHT!"
70 PRINT "THE CORRECT ANSWER IS";X
80 END
```

(2)
```
100 FOR I = 1 TO 12
120 SALES = INT(RND(1)*25001)+25000
140 TSALES = TSALES + SALES
160 NEXT I
180 PRINT "TOTAL SALES", "AVG. SALES"
200 PRINT TSALES, TSALES/12
220 END
```

```
 10 PRINT "ENTER NUMBER OF QUESTIONS"
 20 INPUT Q
 30 FOR I = 1 TO Q
 40 REM CHOOSE ADDITION OR SUBTRACTION
 50 INPUT "CHOOSE A FOR ADDITION, S FOR
    SUBTRACTION.";C$
 60 X = INT (RND(1)*100)+20
 70 Y = INT (RND(1)*50)+25
 80 REM IF ADDITION - GOSUB 160, SUBTRACTION 200
 90 IF C$ = "A" THEN GOSUB 160
100 IF C$ = "S" THEN GOSUB 200
110 PRINT "WHAT IS ";X;A$;Y
120 INPUT AN
130 IF AN = T THEN PRINT "CORRECT"
140 IF AN <> T THEN PRINT "INCORRECT, THE ANSWER IS
    ";T
150 NEXT I
155 END
160 REM ADDITION
170 A$ = "+"
180 T = X+Y
190 RETURN
200 REM SUBTRACTION
210 A$ = "-"
220 T = X-Y
230 RETURN
```

(4)
```
10 REM SIMULATE 2 DICE
20 PRINT "DIE 1", "DIE 2"
30 PRINT
40 FOR I = 1 TO 10
50 D1 = INT(RND(1)*6)+1
60 D2 = INT(RND(1)*6)+1
70 PRINT D1,D2
80 NEXT I
```

(5) What would the following statements display if executed? (Try them at the computer)

```
A.)  PRINT INT(-6.221)
B.)  PRINT INT(8.99)
C.)  PRINT INT(14)
```

(6) What range of values could the following print if executed? (Try each on the computer several times!)

```
A.)  PRINT INT(RND(1)*6)+2
B.)  PRINT INT(RND(1)*20)-20
C.)  PRINT INT(RND(1)*18.6)+0
```

Additional code for program on page 100:

```
 7  N = 0
55  IF ANSWER = L+M THEN PRINT "THAT'S RIGHT!":GOTO 70
62  N = N+1
64  IF N = 2 THEN PRINT "THE CORRECT ANSWER IS ": L+M:
    GOTO 70
68  GOTO 30
```

LESSON 11
AN INTRODUCTION TO ARRAYS

WHAT HAVE YOU LEARNED?

Lesson 11 introduced you to the world of arrays. You learned what an array is and what it does. You also learned how to place information into each element (or memory space) of an array and how to reference individual elements.

Manipulation of arrays also led us into a whole different way to feed information into the computer: the READ-DATA statement. As you saw in Lesson 11, READ-DATA makes it possible to construct programs that run independent of user input and also makes it possible to change elements of data without having to re-input all information.

LET'S TAKE A CLOSER LOOK:

(1) ARRAYS:

An array in BASIC is a collection of similar data accessed by a common variable. For example, if we wanted to keep track of the batting averages of all major league baseball players, all our data would have the same form (that is, all our entries would be 3 digit numbers preceded by a decimal point). Clearly, you would not want to assign each player a separate variable since there may be as many as 1,200 major league players at any given time. If you were to do so, you would first have to look up each player's variable before you could access information for that player. This system would not only be difficult to use, but would actually be easier to arrange without a computer!

Fortunately, Applesoft BASIC has devised a much more effective means of list storage: the array.

When using an array, you must first tell the computer the name of the array and how large it is going to be. (If you did not do so, the computer would not know how much memory space to reserve for this "special" data). The BASIC statement that allows you to do this is the DIM statement. The form of the DIM statement is:

DIM <name> (<number of elements>)

The name of an array can be any legal variable name. The number of elements is the number of separate entries that the

array will have. You may DIMension an array to have extra memory space reserved, although if you leave too much extra space, you will tie up a portion of the computer's memory that may later be needed for your program, but you should never leave too little space, since doing so will halt your program and print an error message.

Some examples of DIM statements are:

```
DIM A$(30) <reserve 30 memory locations in variable A$>
DIM PAYROLL(144) <reserve 144 memory locations in variable
PAYROLL>
DIM SENTENCE$(65) <reserve 65 memory locations in variable
SENTENCE$>
DIM NUMBERS(4)
```

The last example tells the computer to reserve four memory locations in the array called NUMBERS. When the computer processes this statement, it reserves 4 memory cells as below:

```
NUMBERS(1) memory cell 1 of NUMBERS
NUMBERS(2) memory cell 2 of NUMBERS
NUMBERS(3) memory cell 3 of NUMBERS
NUMBERS(4) memory cell 4 of NUMBERS
```

To use (or access) this array you must refer to (or reference) each memory location individually. The number in parentheses in each variable reference is called a subscript. Thus, in speech, we refer to variable NUMBERS(1) as "NUMBERS SUB ONE", A(2) as "A SUB TWO", etc. On the computer, we might make the assignment:

```
LET NUMBERS(1) = 5
```

which tells the computer to place a 5 in the first memory location of the variable NUMBERS.

Can you suggest a way of accessing arrays more easily than by writing statements to access each individual cell?

As you learned in lesson 7, the FOR-NEXT loop is an excellent device for efficiently carrying out repetitive processes.

Let's take a look:

```
100 DIM SALES (12)
110 FOR I = 1 to 12
120 PRINT "ENTER SALES FOR STORE "; I
130 INPUT SALES (I)
140 NEXT I
```

The above lines are used to place yearly sales figures into the array SALES, which has 12 memory locations. When I is equal to 1, the INPUT statement accepts input for SALES(1). When the loop is executed, I becomes 2 and the input accepted is for SALES (2). This continues until I is 12, after which the program moves past this FOR-NEXT loop.

Now that 12 pieces of data have been stored in variable SALES, we can use this data in many different ways. For instance, the lines below allow the user to display the sales for a given store:

```
210  PRINT "ENTER STORE NUMBER"
220  INPUT SN
230  PRINT "SALES FOR STORE ";SN;" WERE "; SALES(SN)
```

by using the store number (SN) as our subscript in the array reference: SALES(SN). Recall that when input was stored in the array, the memory locations were filled in store order. When the computer searches the array, as we are asking it to do in the above example, it "remembers" this order and is able to find the element with the same subscript as the one requested.

It is also possible to have the computer display any stores with sales over, say, $62,000:

```
300  PRINT "THE FOLLOWING STORES HAD SALES GREATER
     THAN $62,000"
310  PRINT
320  FOR I = 1 TO 12
330  IF SALES (I) > 62000 THEN PRINT "STORE "; (I)
340  NEXT I
```

In this example, our FOR-NEXT loop searches array SALES for numbers greater than 62000. Each time it finds one, it prints I, which is serving as our store number. If it does not find one, it simply loops to the Next I.

Suppose, though, you wanted to allow the user to determine the sales limit being seached for; how would you write the code to accomplish this?

```
300  PRINT "ENTER SALES LIMIT"
310  INPUT SL
320  PRINT
330  PRINT "THE FOLLOWING STORES HAD SALES OVER "; SL
340  PRINT
350  FOR I = 1 TO 12
360  IF SALES (I) > SL THEN PRINT "STORE "; I
370  NEXT I
```

In summary, arrays are almost always accessed using FOR-NEXT loops. To return to our baseball example, if we had input the name and batting average of, say, 1,000 baseball players into arrays N$ and BA, we could use the following method of searching for, printing and changing an individual array element:

```
100  FOR J=1 to 1000
110  IF N$(J)="RON CEY" THEN PRINT "PLAYER NUMBER ";
     J,"AVERAGE: ";BA(J)
120  BA(J) = .278
130  NEXT J
```

(2) READ-DATA STATEMENTS

Often when writing a program, we would prefer little or no user intervention. Lesson 11 introduced you to the READ and DATA statements which allow the storage of information inside the program so that when the program is RUN, no user intervention is necessary.

READ and DATA are separate statements that work together. Every time the computer is instructed to READ a variable, it finds the next available DATA line and transfers the first unused piece of data into that variable. Let's look at a sample program to see this process in action:

```
10  DATA 1, 12, 16, 8
20  FOR B = 1 to 4
30  READ A
40  PRINT A
50  NEXT B
```

When this program is executed, it will display the following output:

```
1
12
16
8
```

The READ statement is similar to the INPUT statement, but whereas the INPUT statement obtains its information from the user during the program's execution, the READ statement obtains information from a DATA statement that is present in the program before execution.

When a READ statement is encountered, the program takes the first piece of information from the first DATA statement available. (In the above program ,that is the number 1). Next, it stores that information in the first variable after the READ. If the READ statement has two or more variables (as in READ A, B, C), the next piece of data is placed in the second variable and so on. If another READ statement were encountered, the next available piece of data would be used, and this process would continue until all pieces of data were used.

The computer allows any number of READ and DATA statements and places no limits on the location of DATA statements within the program. The order of the READ statements is important because data is assigned to variables in the order that they are read. However, when a READ statement is encountered, the computer searches for the next available piece of data, so that if you have used two or more lines of DATA statements and one line is used up, the computer will find the next unused one, even if it is in a different part of your program.

Below are several examples of programs using READ and DATA statements.

```
15  FOR I = 1 to 5
20  READ AV
25  TTL = TTL + AV
30  NEXT I
35  DATA 3, 1, 5, 8, 16
40  PRINT "THE AVERAGE OF YOUR NUMBERS IS "; TTL/5
```

OUTPUT:

```
THE AVERAGE OF YOUR NUMBERS IS 6.6
```

Notice that all data are separated by commas. This is how the computer knows that a piece of information is completed.

```
 5  FOR T = 10 to 15
 8  READ V$: PRINT V$
11  NEXT T
14  DATA "A", "P", "P", "L"
17  DATA "E"
```

As you learned, when reading string data, each piece of data should be enclosed in quotation marks and separated by commas.

Initially, we can think of the computer as having an arrow pointing to the first piece of data in the first DATA statement. Looking at the data in the above program, we can visualize this idea as follows:

```
14  DATA "A", "P", "P", "L"
```

When a READ statement is encountered that piece of information is stored in a variable (in the above program, V$). The pointer is then moved to the next item:

```
14  DATA "A", "P", "P", "L"
```

Each READ causes the Data pointer to be moved to the next data item. In this example, after 3 READs, the pointer is at "L". When the next READ is processed, the pointer is moved to the first item of the second DATA statement:

```
90  DATA "E"
```

Suppose we have a reason to re-use the information contained in our DATA statements; can we do it? We may, for instance, want to use the same data set for two entirely different types of analyses during the same program. Is it possible?

Of course it is! The computer provides a mechanism for putting the pointer back to the first item of the first DATA statement. This mechanism is the RESTORE statement. Let's see how it works:

```
10  DATA 6, 7, 8
20  READ X
30  PRINT X
40  READ X
50  PRINT X
60  RESTORE
70  READ X
80  PRINT X
```

The printout looks like this:

```
6
7
6
```

Lines 20 and 30 read and printed the first DATA item, a 6.
Lines 40 and 50 read and printed the second DATA item, a 7.
Line 60 returned the data pointer to the first DATA item.
Lines 70 and 80 read and printed the first DATA item, a 6.

Now that you've completed the first eleven lessons, both on the computer and in this book, you have all the tools that a beginning programmer needs to make the computer do a variety of useful tasks. You can talk to the computer, and you can get the computer to give you the answers you want- if everything goes well. But suppose everything doesn't go well; how will you know?

That's what Lesson 12 is all about!

COMPUTER EXERCISES

1) Write a program to. . .
   A. Store 10 numbers in an array
   B. Search the array for any numbers less than 100 and display them

2) A. Write a program that stores the following table...

	COST
1.	30
2.	80
3.	65
4.	40

   B. Allow a user to enter a number from 1 to 4. Print the corresponding cost in the array.

3) A. Write a program to store the data below in TWO arrays.
   B. For each year, have the program determine which company had higher sales.

	SALES	
YEAR	COMPANY A	COMPANY B
1.	$12,000	$16,000
2.	$ 8,000	$22,000
3.	$23,000	$28,000
4.	$14,000	$30,000

4) What do each of the following display when executed?

A:
```
10 DATA 2,4,6,8
20 READ A
30 PRINT A
```

B:
```
10 DATA "HELLO","HOW","ARE"
20  FOR I = 1 TO 4
30  READ A$
40  PRINT A$:" ";
50  NEXT I
60  DATA "YOU"
```

C:
```
20 FOR I = 1 to 10
30 READ P$
40 PRINTP$
60 NEXT I
70 DATA"1","2"
80 DATA"3","4","5"
```

D:
```
10 READ X$
20 RESTORE
30 PRINT X$
40 FOR I = 1 TO 2
50 READ Y$:PRINT Y$;
60 NEXT I
70 DATA E,L,T,Y
```

ANSWERS-CHAPTER 11

(There is more than one way to skin a cat, and there is also more than one way to write a computer program! The suggestions below will work, but there may be other approaches that work just as well.)

(1)
```
 75  DIM A(10)
 80  FOR COUNTER = 1 TO 10
 85  READ A(COUNTER)
 90  IF A(COUNTER) < 100 THEN PRINT A(COUNTER)
100  NEXT COUNTER
110  DATA 172, 956, 31, 85, 101, 100, 79, 247, 6, 1071
```

(2)
```
120 DIM A(4)
130 FOR J = 1 TO 4
140 READ COST(J)
150 NEXTJ
160 DATA 30, 80, 65, 40
170 INPUT "ENTER A NUMBER FROM 1 TO 4."; NUMBER
180 PRINT NUMBER, COST(NUMBER)
```

(3)
```
200 DIM SA(4), SB(4)
220 FOR YEAR = 1 TO 4
230 READ SA(YEAR), SB(YEAR)
240 IF SA(YEAR) > SB(YEAR) THEN PRINT "YEAR ";YEAR;
    "COMPANYA"; SA(YEAR): GO TO 260
250 PRINT "YEAR: "; YEAR; "COMPANYB "; SB(YEAR)
260 NEXT YEAR
270 DATA 12000, 16000, 8000, 22000, 23000, 28000,
    14000, 30000
```

(4)
```
a:  2
b:  HELLO HOW ARE YOU
c:  1
    2
    3
    4
    5
    ? OUT OF DATA IN LINE 30

d:  E
    EL
```

ADDITIONAL PRACTICE PROGRAMS

1.
A.) Write a program to insert 5 names in an array.

B.) Ask the user to enter a name. If the name is in the array, print YES; otherwise, print NAME NOT IN DIRECTORY.

2. Write a program to print 2 cards from a group of 13 cards.

A.) Store the following cards in an array called CARD$:

A-H	8-H
2-H	9-H
3-H	10-H
4-H	J-H
5-H	Q-H
6-H	K-H
7-H	

B.) Have the computer choose two random numbers:

1.) The first number from 1 to 7
2.) The second number from 8 to 13

C.) Using the two random numbers, display the two cards to which the number refers.

example: If the first random number is 3 and the second random number is 12, the computer should display:

3-H
Q-H

Hint: What card is CARD$(2)?

ADDITIONAL PROGRAMS

SUGGESTED ANSWERS

1.

A.
```
10 DIM A$(5)
20 PRINT "ENTER 5 NAMES"
30 FOR I = 1 TO 5
40 INPUT A$(I)
50 NEXT I
60 PRINT "PLEASE ENTER A NAME."
70 INPUT NA$
80 FOR I = 1 TO 5
90 IF A$(I) = NA$ THEN PRINT "YES":GOTO 120
100 NEXT I
110 PRINT "NAME NOT IN DIRECTORY"
120 END
```

2.

A.
```
10 REM STORE CARDS IN AN ARRAY
20 DIM CARD$(13)
30 FOR J = 2 TO 10
40 CARD$(J) = STR(J) + "-H"
50 NEXT J
60 CARD$(11) = "J-H":CARD$(12) = "Q-H"
70 CARD$(13) = "K-H":CARD$(1) = "A-H"
```

B.
```
80 X = INT(RND(1)*7) + 1
90 Y = INT(RND(1)*6) + 8
```

C.
```
100 PRINT CARD$(X)
110 PRINT CARD$(Y)
```

LESSON 12
ERROR MESSAGES AND DISK I/O

WHAT HAVE YOU LEARNED?

In Lesson 12, you learned the meanings of some of the error messages that the computer prints on the screen when it does not understand your instructions.

You discovered that the computer will give you two clues with each error message that it prints: first, it will tell you the type of error it has discovered and second, it will tell you which line could not be executed because of the error.

Mistakes- or bugs- are extremely common in programming- even among professionals, but the computer is extremely patient; it will run an incorrect program over and over, each time stopping at the point of error and informing you of the problem.

LET'S TAKE A CLOSER LOOK:

(1) SYNTAX ERRORS

By the standards of average human beings, computers aren't very intelligent. True, they have great memories, but like young children, they have to be given each instruction in precise language, or they do not understand it.

In Lesson 11, you saw that while the command "HOME" will clear the screen, the command "ERASE" or the command "HAME" will not. In fact, commands that the computer does not recognize or spelling errors in commands will cause the computer to display the following message:

?SYNTAX ERROR IN <line #>

This is the computer's way of telling you that it doesn't understand something in your program on whichever line it was executing at the time.

Syntax errors may occur for many reasons aside from spelling errors or unrecognized commands. Missing quotation marks, missing or misplaced commas, semi-colons or equal signs and a wide variety of other mistakes in the way in which a line

is written may cause the computer to be unable to proceed. Once the computer has informed you of the type of error and the line number, it has done its job. It then becomes your job to find what the computer has not understood and to correct it.

Let's look at some instructions that will result in syntax errors. Explanations follow the examples, but before you look at them, try to find the error on your own!

```
(A)  PRONT "HELLO"

(B)  PRINT HELLO

(C)  10 A$ = HELLO
     20 PRINT A$

(D)  10 CLEAR SCREEN
     20 PRINT "HELLO"

(E)  10 LET A = 5
     20 LET B = 7
     30 A * B = C
```

Explanations:

	ERROR	CORRECTION
(A)	PRINT IS MISSPELLED	PRINT "HELLO"
(B)	QUOTATION MARKS MISSING	PRINT "HELLO"
(C)	QUOTATION MARKS MISSING	10 A$ = "HELLO"
(D)	UNRECOGNIZABLE INSTRUCTION	10 HOME
(E)	INCORRECT ASSIGNMENT STATEMENT	30 C = A * B

(2) OTHER SOURCES OF ERROR:

Syntax errors aside, Lesson 12 presented several other possible Applesoft BASIC error messages. Let's review them:

(A) ?DIVISION BY ZERO ERROR:

The computer is not equipped to handle a mathematical problem in which division by zero occurs. The statements:

```
10  LET A = 14/0
20  PRINT A
```

will result in the following:

```
?DIVISION BY ZERO ERROR IN 10
```

The same message will result from execution of this program:

```
100  FOR J = -1 to 5
200  K = 5/J
300  PRINT K
400  NEXT J
```

Why? Let's see what happens when this program is executed.

J is set to -1 at the start of the FOR-NEXT loop, so that when line 200 is executed, K is 5 divided by -1. On the second run through the loop, J is zero so that when line 200 is executed K is 5 divided by zero. The computer cannot carry out this calculation, and so the program halts. The following output would result from the execution of this program:

```
-5
?DIVISION BY ZERO ERROR IN 200
```

(B) ?ILLEGAL DIRECT ERROR

This error message will be displayed if you use a deferred mode BASIC programming statement in direct mode.

Simply put, since the direct mode expects to execute each statement immediately, any statement which cannot be immediately executed will generate the ILLEGAL DIRECT ERROR message. Let's look at an example:

```
    DIRECT MODE                  DEFERRED MODE

PRINT "WHAT IS YOUR NAME"   10 PRINT "WHAT IS YOUR NAME"
INPUT A$                    20 INPUT A$
```

The output from these lines would be as follows:

```
    DIRECT MODE                  DEFERRED MODE

WHAT IS YOUR NAME           WHAT IS YOUR NAME
?ILLEGAL DIRECT ERROR       ?
```

In the deferred mode (as part of a program) the computer expects the INPUT statement to be followed by keyboard input. However, in the direct mode, the computer expects to carry out each statement immediately and does not connect statements as in a program. Therefore, the INPUT statement, which requires a keyboard response as a separate statement is not executable in direct mode.

(C) ?ILLEGAL QUANTITY ERROR:

There are many statements used in Applesoft BASIC programming that require numeric arguments. Can you think of some examples?

Let's look at two statements of this type and consider the type of arguments they take:

STATEMENT	THE COMPUTER EXPECTS:
MID$, LEFT$, RIGHT$	A POSITIVE INTEGER TO FOLLOW
ARRAY VARIABLE REFERENCE	A POSITIVE INTEGER

If, for instance you were to RUN the following program:

```
10  LET A$ = "HELLO"
20  PRINT LEFT$(A$, -2)
```

The message printed would be:

```
?ILLEGAL QUANTITY ERROR IN 20
```

because only a positive integer can be used to extract portions of a string using these commands.

Likewise, the following program:

```
10  DIM A(7)
20  FOR J = 1 to 10
30  LET A(J-3) = J
40  NEXT J
```

will result in the same message because when the computer subtracts 3 from J on the first pass through the loop A(J-3) will be A(-2). Since there is no such thing as a negative location, the computer will print the above message.

(D) ?BAD SUBSCRIPT ERROR:

This message is caused by use of an array subscript larger than the array dimension statement has reserved. If, in the previous program, we substitute Line 30:

```
30 LET A(J) = J
```

we will generate this message because when the computer begins the eighth loop for J, A(J) will become A(8). The DIM statement has reserved only 7 memory locations for A and so is faced with a problem it cannot resolve.

(E) ?UNDEF'D STATEMENT ERROR:

This message will appear if you attempt to send the computer to a line in your program that does not exist.

You may generate this message if you use a GOSUB, GOTO or IF-THEN statement. Let's see how:

```
10 A = 5
15 FOR I = 1 TO 3
18 A = A + I
20 IF A > 6 THEN GOTO 28
23 NEXT I
25 END
```

On the second pass through the loop beginning in line 15, A will become 7, and the computer will attempt to transfer the program to Line 28.

Since there is no Line 28, the message

```
?UNDEF'D STATEMENT ERROR IN 20
```

will appear.

(F) ?TYPE MISMATCH ERROR:

Although it is possible in Applesoft BASIC to convert a string to a number (using VAL) and a number to a string (using STR$), it is not possible to mix numerics and strings without using these statements to do so.

The lines:

```
30  A$ = 7
```

will generate:

```
?TYPE MISMATCH ERROR IN 30
```

as will this line:

```
30  A = "HELLO"
```

because Applesoft BASIC does not permit the placing of string data in numeric variables (or of numeric data in string variables without quotation marks.)

This is true not only of assignment statements, but it applies to functions as well. The VAL statement requires a string for conversion, so the statement

```
500  PRINT VAL(28)
```

will result in:

```
?TYPE MISMATCH ERROR IN 500
```

because the computer is expecting VAL to have a string argument and we have given it a numeric argument.

Can you think of other functions that might generate this statement if incorrectly used?

(G) ?OUT OF DATA ERROR:

This common error always results from a READ statement for which you do not have sufficient data in the corresponding DATA statement. Recall from Lesson 9 that when the computer encounters a READ, it finds the first available DATA statement and assigns the first available piece of data to the variable after the READ. But what happens if there is no data left? Let's see:

```
10  READ A
20  PRINT A
30  READ B
40  PRINT B
50  READ C
60  PRINT C
70  DATA 10, 15
```

Output:

```
10
15
? OUT OF DATA IN 50
```

The program has proceeded normally through the first two READ statements, but could not carry out the third because we did not provide it with sufficient data. (Notice that the error message refers to the READ statement even though it is the DATA that is missing.)

(H) ?NEXT WITHOUT FOR:

This error tells you that you haved used a NEXT statement to end a loop but have failed to use a FOR statement to begin it. The following program illustrates this error:

```
100 FOR X = 1TO 5
110 PRINT "HELLO"
120 NEXT X
130 PRINT"GOODBYE"
140 NEXT Y
```

Output:

```
HELLO
HELLO
HELLO
HELLO
HELLO
GOODBYE?NEXT WITHOUT FOR IN 140
```

(I) ?RETURN WITHOUT GOSUB:

This message is similar to the previous one. If the computer encounters a RETURN statement in the body of your program, it attempts to return the program to the line immediately following the GOSUB line that sent it there. If there is no GOSUB line to return to, it prints the above message.

Most commonly, this problem arises when your subroutines are in the body of your program. Since the computer executes the program line by line, it may not only reach a subroutine through a GOSUB statement, but it may also reach it later during the normal execution of that section of your program. When it does, the RETURN statement is meaningless and the error message is generated. We suggest, therefore, that you separate your subroutines and place them after the END statement of your program.

We've come to the end of Lesson 12- and The Basics of BASIC. We've given you a foundation in BASIC programming- but that foundation needs to be built upon- with even more challenging practical application.

We wish you luck and patience, and don't forget, the Basics of BASIC is always here to come back to if you need a refresher along the way!

COMPUTER EXERCISES

For each of the following programs, explain what is wrong and the error messages that are likely to occur if the programs are executed.

1)
```
10 DIM B(20)
20 FOR I = 1 TO 100
30 B(I) = I
40 NEXT I
```

2)
```
10 PRINT "ENTER A NUMBER"
20 INPUT N
30 PRINT  "1000 DIVIDED
   BY ";N; " IS ";1000/N
```

3)
```
INPUT "ENTER A NUMBER";N
```

4)
```
10 PRINT MID$
("IMPOSSIBLE",-2,11)
```

5)
```
10 PRINT "ENTER A NUMBER"
20 GOTO 40
30 END
40 INPUT N
50 RETURN
```

6)
```
10 FOR I = 1 TO 260
20 X$ = X$ + I
30 NEXT I
```

7)
```
10 PRINT "ENTER SPEED"
20 INPUT SP
30 IF SP = "100" THEN PRINT "FAST"
```

8)
```
10 PRINT ((4*(3+2/8+9)-(16/2)
```

ANSWERS-CHAPTER 12

(1) This program will produce a BAD SUBSCRIPT error because the DIM statement has left room for 20 items in array B and we are attempting to place 100 items into the array.

(2) Here we may have an attempt to carry out DIVISION BY ZERO. Consider how you might build in a subroutine to check the INPUT and reject a zero as input.

(3) This line will produce an ILLEGAL DIRECT ERROR since INPUT must be used in a program (deferred execution mode).

(4) This line will produce an ILLEGAL QUANTITY ERROR because we have used a negative number with the MID$ function, which only works with positive numbers.

(5) This is an attempt to use a RETURN WITHOUT GOSUB.

(6) Here we have created a TYPE MISMATCH by attempting to add a numeric to a string.

(7) Again we have a TYPE MISMATCH since SP is a numeric and we have tried to compare it to a string.

(8) This is a SYNTAX ERROR because we have 4 opening parentheses but only two closing. The computer cannot evaluate the expressions within unless there is a closing parenthesis for every opening parenthesis.

APPENDIX A
HARDWARE: COMPONENTS OF THE APPLE COMPUTER

The term HARDWARE is used to denote the PHYSICAL COMPONENTS of your computer. Any part of the computer that you can touch is considered hardware.

We will organize our discussion of your Apple computer by breaking up the hardware into three categories:

1) INPUT/OUTPUT DEVICES
2) PROCESSOR UNIT
3) MEMORY

1) INPUT/OUTPUT DEVICES:

Input/output devices are used to communicate with the computer. Parts of your computer that are included in this category are the keyboard, the monitor, and the disk drive(s).

The keyboard is your way of telling the computer what to do. It allows you to enter programs and information which the computer can then process. Therefore, it provides a mechanism for you to communicate with the computer.

The monitor, on the other hand, allows the computer to communicate with you. The computer responds to your commands by displaying information on the screen. Also, any data that you enter on the keyboard is seen on the monitor. The monitor can be thought of as a special type of television screen.

Disk drives are used to save programs and data. Just like a cassette recorder is used to store songs, disk drives enable the computer to store information on floppy disks. This information may be retrieved again and again, just like a song can be played more than once.

2) THE PROCESSOR UNIT:

Although the computer must be told what to do, it does have the ability to interpret information. Its processor unit has the capability of controlling both the input/output devices attached to it and its memory storage. You may think of the processor unit as the "brains" of your computer system. It is the part of

your computer that performs calculations and prints out answers to questions.

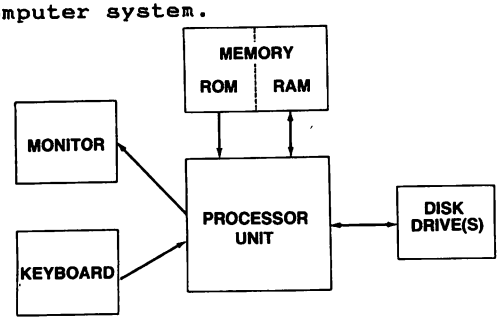
3) MEMORY

The processor unit works in conjuction with the memory of your computer. One of the reasons why computers are so useful is their ability to store information. Memory may be visualized as many little containers, each capable of holding a piece of data. The memory in the Apple is broken up into two parts:

1) RAM (Random access memory)
2) ROM (Read-only memory)

Both RAM and ROM can contain data. However, there are key differences between the two. RAM is memory that can be changed, and is erased when the computer is turned off. When the processor unit performs a calculation it stores its answer in the RAM. If there was already information stored in the RAM it may be erased. The memory in ROM can never be changed. It contains the information which tells your computer how to operate. ROM can be thought of as a phonograph record. The record has songs stored on it which cannot be changed or modified. Whenever you play the record the same songs will be heard; they cannot be erased.

The following diagram describes the flow of information within your computer system.



Notice that the processor unit is in the center of the diagram, because it controls all of the other components of the computer. For this reason it is sometimes referred to as the Central Processing Unit (or CPU). Also notice which way the arrows are pointing. As you can see, the central processing unit can send information to the monitor and receive information from the keyboard. The arrow pointing from ROM to the processor unit indicates that the processor unit can access ROM but cannot store anything in ROM. The double arrows indicate that the computer can both send and receive information.

APPENDIX B
PREPARING A WORK DISK

During the course of these lessons, you may want to place programs on your own diskette for easy reference or use. If you have a new diskette, before it can be used it must be FORMATTED following the instructions below.

A. Your Apple comes with a system master (sometimes called DOS for disk operating system) diskette. Place this diskette in drive 1 and turn on the power.

B. When the red light on the front of the disk drive goes off, remove the system master from the drive. NOTE: NEVER OPEN THE DOOR OF A DISK DRIVE WHILE THE RED LIGHT IS ON!

C. Place a blank diskette in the drive and close the drive door.

D. Type the following lines exactly as you see them below. Press the RETURN key after typing each line:

```
NEW
10 PRINT "WORK DISK"
INIT HELLO
```

E. Remove the diskette from the drive. Label it as your work disk either on its label or on the jacket.

GLOSSARY
IMPORTANT WORDS AND BASIC COMMANDS

(After each definition, the chapter in which where the word, phrase or command is first introduced appears in parentheses.)

Argument: the constant or expression used by a function. (10)

Array: a table in which the elements share a common variable name and are referenced using that variable name accompanied by a subscript in parentheses. (11)

Assignment: the placing of a value or string in a variable. (3)

Booting: the process of adding the operating instructions to the computer system, either using a disk specifically designed for the purpose or by using software that is self-booting. (Introduction)

Branching: the process of transferring control within a program from one line or section to another. (7)

Bug: an error in either the logic or the syntax of a program. (12)

CATALOG: the instruction that causes all the files on the diskette in the specified drive to be displayed. (If no drive is specified or if only one drive is present, then the command will display the directory of the disk in drive 1.) (5)

Concatenation: the combining of strings by adding them together. (3)

CTRL/C: pressing the 'C' key while holding down the CTRL key will abort the running or listing of a program. (Getting Ready)

CTRL/S: depressing the CTRL and 'S' key simultaneously while listing a program will halt the listing at that point. Pressing any other key will cause the listing to resume. (4)

Cursor: the blinking square that appears on the screen to prompt the input of information.

DATA: the DATA statement provides the data that is sequentially assigned to each variable in a READ statement. (11)

DEL 100,200: the statement that removes the specified range of lines from a program. (The above statement would remove lines 100 through 200 inclusive from the program.) (5)

DELETE PROG1: the statement that deletes a file from the diskette in the specified drive. (If no drive is specified, then drive 1 is assumed.) (5)

DIM STATE$(50): the statement that sets aside space within an array. (11)

DOS: the disk operating system, or system of instructions that enables the computer to work. (5)

END: the statement that causes a program to stop execution.(4)

FLASH: the statement that causes output following to be shown alternately in black characters on a white screen and white characters on a black screen. (6)

FOR: the initial statement in a FOR-NEXT loop. (Example: FOR J = 1 TO 20 STEP 5) (7)

Formatting: the process of arranging for output to appear in an appropriate form.

Function: Anything in BASIC that accepts one or more numbers or variables, performs an operation or operations on them, and yields a single value as a result of those operations. (10)

GOSUB: the statement that causes a program to branch to an indicated line and execute each successive line until a RETURN statement is encountered. (7)

GOTO: the statement that causes a program to branch to the indicated line and continue execution at that point. (7)

HOME: the statement that moves the cursor to the upper lefthand corner of the screen and clears the screen of all text. (5)

HTAB: the statement that moves the cursor to the specified column on the screen. (6)

IF: the statement that causes the computer to evaluate an expression in order to make a decision. (Example: IF A = 3 THEN PRINT "VERY GOOD!") (8)

INIT HELLO: the statement that initializes a diskette. Use of this statement with a diskette that already has files on it will have the effect of erasing them.(10)

INPUT: the statement that causes the computer to assign a variable to the user input that follows. (5)

INT (47.87): the statement that returns the largest integer less than or equal to its argument. (10)

INVERSE: the statement that sets the video mode so that the screen output prints as black letters on a white background. (6)

LEFT$ (ALLAN,2): the statement that returns the specified number of leftmost characters from a string. (9)

LEN ("JOSEPH"): the statement that returns the number of characters in a string. (9)

LET: the statement that causes the assignment of a number or string to a variable. (3)

LIST: the statement that causes the specified portion of a program to be displayed on the screen. If no line numbers are specified, the entire program will be displayed. (4)

Literal: a word or phrase in a program that appears in quotes and that will appear in the output exactly as written. (2)

Loop: a programming sequence that is repeated either a specified number of times or until the specified exit condition is met. (7)

MID$ (A$,X): the statement that returns the specified substring beginning with the first character indicated from the left and continuing until X characters have been returned. (9)

NEW: the statement that deletes the current program from memory and clears all variables. (4)

NEXT: the closing statement of a FOR-NEXT loop. (7)

NORMAL: the statement that sets the video mode to the usual white letters on a black background. (6)

PRINT: the statement that causes the specified information to be printed. (2)

RAM: short for Random Access Memory; the section in the computer's memory that is used for program storage and manipulation. RAM is volatile (that is, it is erased when its electrical power supply is interrupted). (3)

READ (X,Y): the statement that instructs the computer to assign the next available items of DATA to the specified variables. (11)

REM: the statement that allows text to be inserted into a program as remarks. (4)

Reserved words: words that are considered by the computer to be programming or DOS instructions. These words cannot be used as variable names in programs. (3)

RESTORE: the statement that causes the data pointer to return to the first item of data in the first DATA statement before making another assignment to a READ variable. (11)

RETURN: the statement that returns the program to the statement immediately following the last GOSUB to have been executed. (7)

RIGHT$ ("APPLE",3): returns the specified number of rightmost characters from the string (in this case, PLE). (9)

RND(1): the statement that returns a random number between 0 and 1. (10)

ROM: short for Read Only Memory; that part of the computer's memory that controls the computer's operating system. ROM is non-volatile (that is, it will retain information regardless of the state of the computer's power supply). (3)

RUN: the statement that begins execution of a program at the specified line. If no line is specified, execution will begin with the first line of the program. (4)

SAVE PROG1: the statement that stores the program currently in memory on the disk in the specified drive. (5)

STR$ (308): the statement that converts a numeric argument to its string equivalent. (9)

Subroutine: that portion of a program that begins with the first line called by a GOSUB statement and ends with the next RETURN statement encountered. (7).

Syntax: the structure, spelling or grammar of a BASIC statement. (3)

TAB (5): the statement which, used in conjunction with a PRINT statement, causes printing to begin in the specified column. (6)

THEN: the resultant action of an IF expression. (8)

Truncation: the removal of the decimal point and all following digits in a numeric expression. (10)

User-friendly: an expression that describes the extent to which a program is easy to understand and easy to use. (5)

VAL ("752"): the statement that converts a string to its numeric equivalent. If the string does not begin with a number, the value returned will be 0. If the string begins with a number but has non-numeric characters the value returned will include all the characters, up to the first non-numeric character. (9)

Variable: the name given in a program to a location within the computer that will hold information. (3)

VTAB (14): the statement that moves the cursor to the line on the screen specified by the argument. (6)